AVİYONİK ÜNİTELERİN SİMULATÖR SİSTEMLERİ İÇİN MODELLENMESİ

Berna KARAMAN
Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Eylül – 2004

Modellenmesi başlıklı **Bilgisayar Mühendisliği** Anabilim Dalındaki, Yüksek Lisans tezi 01 Eylül 2004 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	İmza
Üye (Tez Danışmanı)	**: Doç. Dr. Ahmet BABANLI**	
Üye	**: Prof. Dr. Ali GÜNEŞ**	
Üye	**: Yrd. Doç. Dr. Ayşe KAHVECİOĞLU**	

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 08.09.2004. **tarih ve** 29/22 **sayılı kararıyla onaylanmıştır.**

Enstitü Müdürü
Prof. Dr. Altuğ İFTAR
Fen Bilimleri Enstitüsü
Müdürü

ÖZET

Yüksek Lisans Tezi

AVİYONİK ÜNİTELERİN SİMULATÖR SİSTEMLERİ İÇİN MODELLENMESİ

BERNA KARAMAN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç.Dr.Ahmet BABANLI
2004, 149 sayfa

Bu tezde, aviyonik ünitelerin simulatör sistemleri için modellenmeleri ele alınmıştır. Ancak tüm aviyonik ünitelerin modellenmesi değil, sadece gerçek şartlarının sağlanması mümkün olmayan aviyonik ünitelerin modellenmeleri irdelenmiştir.

Simulatör sistemlerinin, simulasyonu gerçekleştirilecek sistemin gerçek hayatta gerçekleştirdiği her türlü fonksiyonu karşılayabilmesi gerekmektedir. Uçan platformlarda bulunan aviyonik ünitelerin bir kısmı uçuş sırasında, uçan platformun bulunduğu konuma, yaptığı hareketlere göre gerekli çıktıları oluştururlar. İşte bu şekilde çalışmakta olan aviyonik ünitelerin gerçeklerinin simulatörlerde kullanılmaları mümkün olmamakta ve bu sebeple modellerinin oluşturulması ihtiyacı doğmaktadır. Modellenme ihtiyacı bulunmayan üniteler önemli modifikasyon gerektirmeden simulatör sistemlerinde kullanılabilmektedir. Dolayısıyla bu tezde simülatör sistemlerinde olduğu gibi kullanılamayacak aviyonik ünitelerin modellenmeleri anlatılmıştır.

Anahtar Kelimeler: Modelleme, Aviyonik, Seyrüsefer, Tanımlama, C Programlama

ABSTRACT

Master of Science Thesis

MODELING OF AVIONIC UNITS FOR SIMULATOR SYSTEMS

BERNA KARAMAN

Anadolu University
Graduate School of Natural and Applied Sciences
ComputerEngineering Program

Supervisor: Prof.Ahmet BABANLI
2004, 149 pages

In this thesis the modeling of the avionic units is tried to be explained. However, it is not the modeling of all types of avionic units, but only the units which can not be used in simulator systems because of their operating environment can not be simulated.

Simulator systems should support all the functions of the systems, to give the feeling of reality. For the operation of some of the avionic units, aircrafts air condition properties are needed. The attitudes of the aircraft, the environment data around the aircraft are the examples of the needed inputs to such avionics systems. These kind of conditions can not be performed in simulators, so the models of these systems are needed. The systems that the modeling is not needed can be used in simulator systems without any modification. For the systems which do not need the modification, can operate by feeding its inputs. So in this thesis, the avionics systems which cannot be operated in simulator sytems are going to be observed and their modeling ways are going to be told.

Keywords: Modeling, Avionics, Navigation, Identification, C Programming

TEŞEKKÜR

Öğrencisi olarak kendisiyle çalışmaktan büyük onur duyduğum hocam Doç. Dr. Ahmet BABANLI'ya, çalışmalarım sırasında beni yönlendirdiği için kendisine teşekkürü bir borç bilirim.

Ayrıca desteğini hiçbir zaman esirgemeyen Aileme, Bora BARIN'a, Abdülaziz BEKKİNE'ye ve Hakkı Ulaş ÜNAL'a teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER ve KISALTMALAR DİZİNİ

A/A	Air to Air
A/D	Analog to digital
A/G	Air to Ground
AC	Aircraft
ADC	Air Data Computer
AOA	Angle of Attack
BBC	Backup Bus Controller
BC	Bus Controller
BM	Bus Monitor
BPSK	Binary Phase Shift Keying
CAS	Calibrated Airspeed
C/A	Coarse/Acquisition
CDI	Course Deviation Indicator
CDMA	Code Division Multiple Access
CIG	Computer Image Generator
COESA	Committee on Extension to the Standard Atmosphere
D/A	Digital to Analog
DME	Distance Measurement Equipment
EGI	Embedded GPS INS
EMRG	Emergency
FM	Frequency Modulation
FOV	Field of View
GHz	Giga hertz

GPS	Global Positioning System
HUD	Head Up Display
HSI	Horizontal Situation Display
ICD	Interface Control Document
ICS	Intercommunication System
IFF	Identification Friend or Foe
IG	Image Generator
INS	Inertial Navigation System
IR	Inertial Reference
LSB	Least Significant Bit
MFD	Multi Function Display
MHz	Mega hertz
MIL-STD	Military Standart
MSB	Most Significant Bit
NASA	National Aeronautics and Space Administration
NATO	National Atlantic Treaty Organization
NM	Neutical Miles
NOAA	National Oceanic and Atmospheric Administration
PC	Personal Computer
PRN	Pseudo Random Noise
RALT	Radar Altimetre
REC	Receive
RT	Remote Terminal
SAE	Society of Automotive Engineers
SIF	Selective Identification Feature

STBY	Standby
TACAN	Tactical Air Navigation
TAS	True Air Speed
TAT	Total Air Temperature
TOD	Time of Day
T/R	Transmit Receive
UHF	Ultra High Frequency
U.S.	United States
VHF	Very High Frequency
VME	Versa Module Eurocard
VOR	VHF Omnidirectional Range
WOD	Word of Day
WP	Way Point

1. GİRİŞ

Günümüzde pilotlar; elektronik cihazlarla dolu bir ortama, elektronik cihazları seri olarak kullanabilmeye ve uçuş esnasındaki her türlü taktik ve acil duruma hazır olabilmek için iyi eğitilmiş olmalıdırlar.

Eğitimin gerçek uçaklarla yapılması hem maliyeti arttırır, hem de istenilen acil durum eğitimi yapılamaz. Gerçek uçakla yapılması tehlikeli olan, inişte motor arızası, açılmayan iniş takımı, jeneratörün kaybı gibi acil durumların eğitiminin yeri pilot eğitimlerinde oldukça büyüktür. Bu eğitimlerin güvenli bir şekilde yapılabilmesi için gerçek uçakların yerini, aynı orjinallikte olan "simülatör' uçaklar almıştır ve bugün her türlü uçak eğitimlerini bu simülatörlerde güvenli bir şekilde gerçekleştirebilmektedir.

Simulatörler ilk tasralandıklarında, uçak üzerinde bulunan bazı ünite ve techizatın gerçek çalışma şartları sağlanamadığı için gerçeklenemediği görülmüştür. Gerçek şartlardaki çalışma prensiplerini yerine getirebilmek için bu ünitelerin mekanik, elektronik ve/veya yazılımsal olarak modellenmesine ihtiyaç duyulmuştur.

Günümüz uçaklarının mevcut sistemlerinden biri olan aviyonik sistemler genel olarak uçağın mevcut konumunu belirleyen ve uçuş malumat bilgilerini üreten seyrüsefer sistemi, pilotların dış dünya ile iletişimini sağlayan haberleşme ve dost/düşman tanıma sistemi, ve tüm bilgilerin pilot için görüntülendiği gösterge kontrol sistemlerinden meydana gelmektedir.

„Aviyonik" (avionics) kelimesi, havacılık (aviation) ve elektronik (electronics) kelimelerinin birleşmesinden meydana gelmiş ve tam olarak havacılık elektroniği anlamına gelmektedir. Aviyonik sistemler incelendiğinde gösterge ve kontrol sistem üniteleri olan göstergeler, video kayıt cihazları, video kameralar ve bunların kontrolunu sağlayan kontrol üniteleri hariç seyrüsefer sistemleri olan ataletsel seyrüsefer sistemi, taktik hava seyrüsefer sistemi, hava veri bilgisayarı ve yükseklik ölçer ile iletişim sistemleri olan radyo ve dost/düşman tanıma sistemleri gibi diğer aviyonik sistemlerin simulatör şartlarında çalışmadıkları ve bunların modellenmelerine ihtiyaç duyulduğu görülmüştür. Bu ünitelerin çalışmalarını gerçekleyebilmek için yazılım

modellerini geliştirme ihtiyacı doğmuştur. Bu nedenle bu tezde simulatör sistemlerinin ihtiyacını karşılayacak şekilde aviyonik ünitelerin modellenmesi üzerine çalışılmıştır.

Modellemenin iyi bir şekilde gerçekleştirilebilmesi için öncelikle modellenecek sistemin gerçek çalışma şartlarındaki her türlü çalışma prensibinin ortaya konması, daha sonra da modelin entegre edileceği sistemin yapısının belirlenmesi gerekmektedir. Bu bilgilerin iyi kavranaması, modelin ihtiyacı olan girdi bilgilerinin elde edilebilmesi ve model çıktılarının ihtiyacı olan sistemlere ulaştırılabilmesi için önemlidir.

Her bir ünitenin çalışma prensibine göre kullanılacağı simulatör sisteminde modellenmesi farklılık gösterebilmektedir. Bu farklılıklar kullanıcı isteğine göre şekillenmektedir. Bu nedenle bahsedilen modelin ana çalışma prensiplerinin modellenmeleri bu tezde irdelenmiştir. Her bir aviyonik ünitenin diğer sistemlerle olan arayüzü de sistemden sisteme değiştiğinden arayüz özelliklerinden de bu tezden faydalanan kişilerin bilgi edinebilmesi için genel olarak bahsedilmiştir.

Bölüm2'de, modellenmesi anlatılacak olan Hava Veri Bilgisayarı, Yükseklikölçer, Taktik Hava Seyrüseferi ve Dost Düşman Tanıma sistemlerinin temel çalışma prensiplerinden, yapılan modelin daha anlaşılır olması için bahsedilecektir. Modellenmek üzere yukarıda bahsedilen ünitelerin seçilmelerinin sebebi, simulatör sistemlerinde uygulanan modelleme çeşitlerinden daha fazla bahsedebilmektir. Tüm bahsedilen üniteler incelendiğinde simulatörlerde yapılan genel modelleme yöntemleri etraflıca incelenmiş olacaktır.

Bölüm 3'de aviyonik sistemlerin uçak sistemleri ile olan ilişkisi ve buna göre modelde sahip olacağı yeri kavrayabilmek için genel olarak uçak sistemleri ve simulatör sistemlerinin yapısından bahsedilmiştir. Bu sayede gerçekleştirilen modellerin ihtiyacı olan girdi bilgilerini elde edebilecekleri diğer sistemler daha kolay kavranabilecektir.

Bölüm 4'de simulasyon modellerinin hazırlanması ve yazılımından bahsedilmiştir. Ünitelerin modellerinin hazırlanması sırasında dikkat edilen hususlar ve örnek modellerin tasarımları ve matematik modelleri anlatılmıştır. Ancak modellerin arayüz iletişimlerinden genel olarak bahsedilmiştir. Bunun

sebebi projesi gerçekleştirilen simulatörden simulatöre arayüz özelliği değişim gösterebilmesidir.

Bölüm 5'de sonuçlar ve tartışma kısmında ise hazırlanan modellerin hazırlanma şartları, hazırlanma sırasında karşılaşılan problemlerden bahsedilmiştir.

2. AVİYONİK ÜNİTELERİN ÖZELLİKLERİ

Bu bölümde, modellemesi üzerinde çalışılmış olan hava veri bilgisayarı, yükseklikölçer, Taktik hava seyrüseferi ve dost/düşman tanıma sistemlerinin gerçek şartlardaki çalışma prensiplerinden bahsedilmiştir. Bu ünitelerin modellenmelerinin gerçeklenebilmesi için önce çalışma prensiplerinin kavranması gerekmektedir. Böylelikle, her bir ünitenin kullanım amaçları, ünitelerin ihtiyacı olan girdiler ve ünitelerden beklenen çıktılar, girdilerden çıktılara ulaşabilmek için yapılan işlemler ve ünitelerin çalışma teoremlerinden bahsedilecektir.

2.1.Hava Veri Bilgisayarı

Uçuş veri sistemi, aerodinamik ve termodinamik sensörler ile elektronik ünitelerden oluşan, uçağın uçuş esnasında ihtiyacı olan bilgileri üreten seyrüsefer sistemidir. Sensörler, uçağın etrafını çevreleyen havanın karakteristiğini ölçerek bu bilgileri transducer birimleri vasıtasıyla elektrik sinyallerine dönüştürürler. Elde edilen sinyaller hava veri bilgisayarında işlenir ve böylece uçuş parametreleri elde edilmiş olur.

Buna göre tipik uçuş parametreleri,

- kalibre edilmiş hava hızı (calibrated air speed),
- gerçek hava hızı (true air speed),
- mach sayısı,
- basınç yüksekliği,
- barometrik yükseklik,
- statik hava sıcaklığı,
- hava yoğunluğu,
- hücum açısı
- yan-kayma (side-slip) açılarıdır.

Bu parametreler uçuş göstergeleri için, uçağın otopilot sistemi için, seyrüsefer sistemi için (EGI – Embedded GPS/INS), silah sistemi için ve sivil uçaklarda kabin-hava basınç kontrol sistemi için kullanılırlar [1].

Aviyonik sistemlerde uçuş parametrelerini üreten sistem Air Data Computer (ADC)'dir. Uçakta bulunan uçuş bilgi sensörlerinden aldığı verileri kullanarak uçuş parametrelerini hesaplar. Üretilen bu bilgiler çeşitli aviyonik göstergelerde görüntülenirler ve diğer sistemlerin faaliyetlerini gerçekleştirmeleri için ilgili sistemlere aktarılırlar.

2.1.1.Hava veri bilgisayarının girdi bilgilerinin elde edilmesi

Uçuş performansı ile ilgili tüm uçuş veri parametreleri, uçağın etrafındaki basınç, sıcaklık ve akış yönünün sensörler tarafından hissedilmesi ile elde edilirler. Bu girdi bilgileri;

- statik portlardan alınan statik basınç bilgisi, P_{si},
- pitot tüpünden alınan toplam basınç bilgisi, P_{ti},
- sıcaklık probundan alınan toplam sıcaklık bilgisi T_{ti},
- AOA sensöründen alınan akış yönü/hücum açısı, (αL), bilgisidir [1].

2.1.1.1. Statik basınç

Statik basınç, uçağın çevresinde bulunan gerçek basınçtır. Statik portlardaki delikler vasıtasıyla hareket eden uçağa statik hava alınır. Bu hava statik port aerodinamik ünitesince statik basınç bilgisi haline getirilir [1].

Statik basınç P_s

$$P_s = P_{si} \cdot C_f \qquad (2\text{-}1)$$

formulüyle verilmektedir.

Burada C_f statik kaynak düzeltme faktörü olup P_{si} değerine göre 0.181 ile 1.600 arasında değer almaktadır.

2.1.1.2. Toplam basınç

Durgun basınç veya pitot basıncı da denilen toplam basınç, uçağın pitot tüpünden alınan havanın pnömatik birimlerce hissettiği basınçtır [1].

Toplam basınç P_t

$$P_t = P_{ti} \tag{2-2}$$

formulüyle verilmektedir.

2.1.1.3. Toplam sıcaklık

Uçağın sıcaklık probunun direnci, uçak havada hareket ettikçe oluşan ısı değişimi değerini hissedecek yapıda ısıya duyarlı bir dirençtir. Bu direnç değerinin değişimi sonucunda uçağın dışındaki sıcaklık hissedilir [1].

Toplam sıcaklık direnci (R_T)

$$R_T = R_O + \alpha \cdot R_O \cdot \left[T_{ti} - \gamma \cdot (T_{ti}/100 - 1) \cdot (T_{ti}/100) - \beta \cdot (T_{ti}/100 - 1) \cdot (T_{ti}/100)^3 \right] \tag{2-3}$$

formulü ile hesaplanmaktadır[1].

Burada R_T toplam sıcaklık direncini,

R_0, 0°deki direnç (50Ω);

α toplam sıcaklık prob sabiti (0.003832);

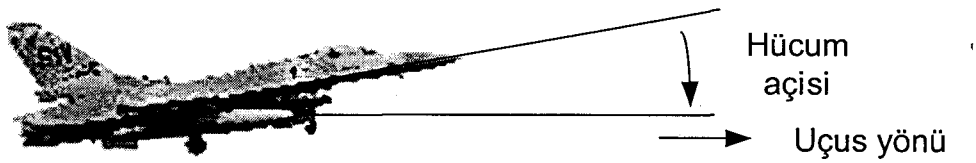
γ toplam sıcaklık prob sabiti (1.82);

β toplam sıcaklık prob sabiti ($T_{ti} \geq 0°C$ ise $\beta = 0$, $T_{ti} < 0°C$ ise $\beta = 0.1$) olarak tanımlanmaktadır.

R_T, R_o, α, γ, β değerleri (2-3) eşitliğinde yerlerine konulduğunda toplam sıcaklık değeri (T_{ti}) elde edilir [1].

2.1.1.4. Hücum açısı

Hücum açısı, uçağın yönü doğrultusundaki yatayla yaptığı açıdır [2]. Hücum açısı Şekil 2.1'de verildiği şekildedir.



Şekil 2.1 Hücum açısı

Hücum açısı, uçak üzerinde bulunan Angle of Attack (AOA) vanasındaki direnç değişiminin, atak açısı sıfır değerinde iken olan direnç değerine oranı belirlenerek tespit edilir [1].

Buna göre hücum açısı (αL) bilgisi [1]

$$\alpha L = 55 \cdot R\alpha - 12.5 \qquad (2\text{-}4)$$

formülü ile hesaplanmaktadır.

Burada $R\alpha = R_i \cdot 2750$;

R_i, hücum açısı vanası direnci,

2750 ise hücum açısı 0 olduğunda elde edilen dirençtir.

2.1.2.Hava veri bilgisayarının yaptığı hesaplamalar

Hava veri bilgisayarı yukarıda bahsedilen girdi bilgilerinin elde ettikten sonra çıktı verilerini elde etmek için çeşitli hesaplamalar yapmaktadır. Uçuş parametreleri olarak adlandırılan bu verilerin hava veri bilgisayarı içerisindeki hesaplama yöntemi aşağıda anlatılmıştır.

2.1.2.1. Kalibre edilmiş hız

Hava veri bilgisayarı kalibre edilmiş hızı hesaplayabilmek için statik ve toplam basınç değerlerinden etki basıncı Q_c, ve görüntülenen etki basıncını Q_{ci} hesaplamaktadır [1].

Buna göre etki basıncı Q_c

$$Q_c = P_t - P_s \qquad (2\text{-}5)$$

görüntülenen etki basıncı Q_{ci}

$$Q_{ci} = P_t - P_{si} \qquad (2\text{-}6)$$

formülleri ile hesaplanmaktadır.

Kalibre edilmiş hız (V_C),

$$Q_c = P_{so} \cdot \left[\left(1 + 0.2 \cdot [V_c / C_o]^2\right)^{3.5} - 1\right] \qquad V_c < C_o$$

$$Q_c = P_{so} \cdot \left[\left[166.9216 \cdot (V_c / C_o)^7\right] / \left[7 \cdot (V_c / C_o)^2 - 1\right]^{2.5}\right] \qquad V_c \geq C_o \qquad (2\text{-}7)$$

eşitliklerinden elde edilmektedir.

Burada P_{so} deniz seviyesindeki statik basınç olup 29.9213,

C_o ise deniz seviyesindeki ses hızıdır olup 661.4786 knots olarak sabittir [1].

2.1.2.2. Mach sayısı

Mach sayısı (M_n)

$$P_s / P_t = \left(1 - 0.2 \cdot M_n{}^2\right)^{-3.5} \qquad M_n \leq 1 \qquad (2\text{-}8)$$

$$P_s / P_t = \left(1/1.2 \cdot M_n{}^2\right) \cdot \left[\left(7 \cdot M_n{}^2 - 1\right) / \left(7.2 \cdot M_n{}^2\right)\right]^{2.5} \qquad M_n > 1 \qquad (2\text{-}9)$$

eşitliklerinden elde edilmektedir.

2.1.2.3. Gerçek hız

Gerçek hız (V_t),

$$V_t = 29.045 \cdot M_n \cdot (T_{FAT})^{0.5} \qquad (2\text{-}10)$$

eşitliğinden elde edilmektedir [1].

2.1.2.4. Basınç yüksekliği

Basınç yüksekliği (H_p),

$$H_p = 145442 \cdot \left[1 - (P_s / 29.9213)^{0.19026}\right] \qquad H_p < 36089\,ft \qquad (2\text{-}11\text{-a})$$

$$H_p = 36089 - 20806 \cdot [\ln(P_s / 6.68324)]$$

$$36806 ft \le H_p < 65617 ft \qquad (2\text{–}11\text{-b})$$

$$H_p = 65617 - 710794 \cdot \left[1 - (P_s / 1.61673)^{0.029271}\right]$$

$$65617 ft \le H_p < 104987 ft \qquad (2\text{–}11\text{-c})$$

eşitliklerinden elde edilmektedir.

2.1.2.5. Basınç yüksekliği

Barometrik düzeltilmiş yükseklik (H_{bc}),

$$H_{bc} = H_p(P_s) - H_p(PBS) \qquad (2\text{-}12)$$

eşitliğinden elde edilmektedir.

2.1.2.6. Tırmanma oranı

Tırmanma oranı (R_c)

$$R_c = d(H_p)/dt \qquad (2\text{-}13)$$

eşitliğinden elde edilmektedir.

2.1.2.7. Toplam sıcaklık

Toplam sıcaklık (T_t)

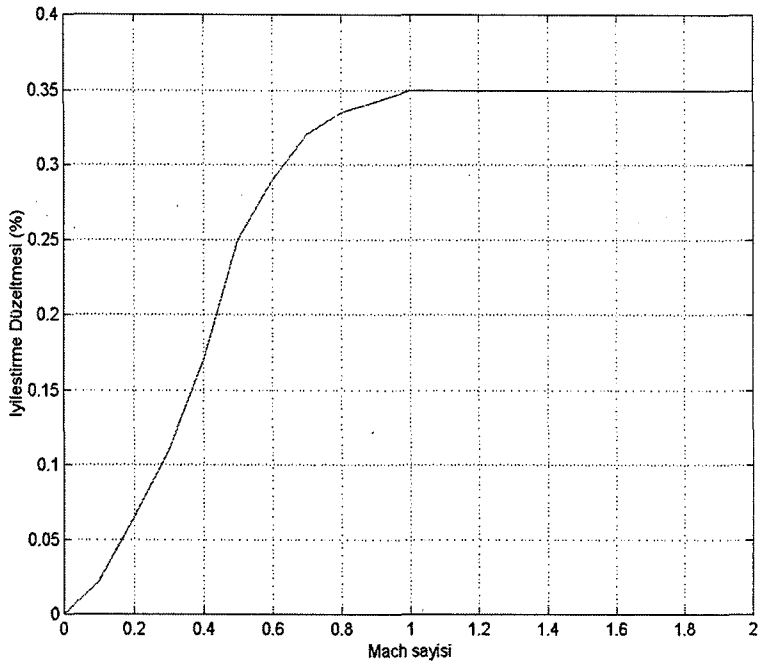
$$T_t = T_{ti} / (1 - \eta) - DHE \qquad (2\text{-}14)$$

eşitliğinden elde edilmektedir.

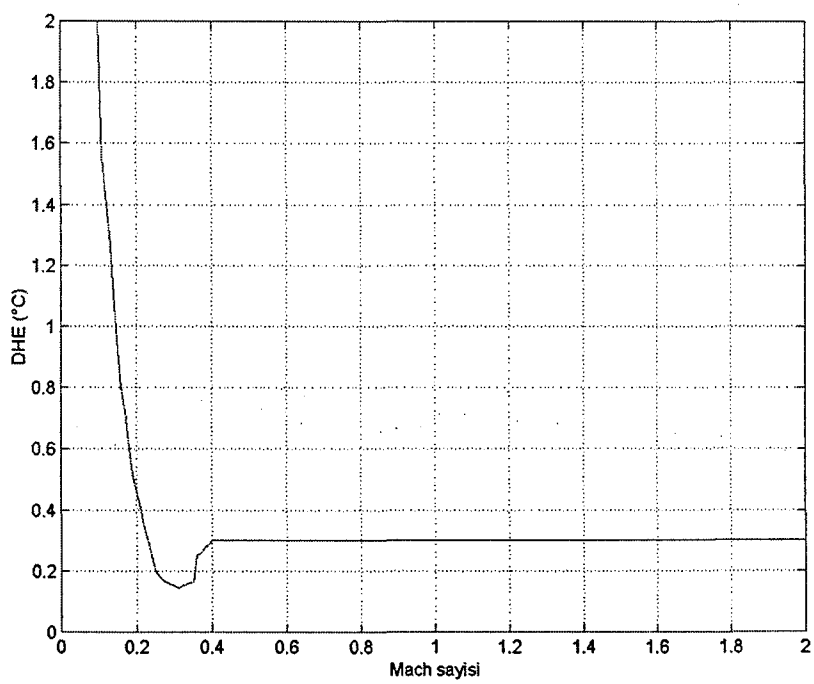
Burada DHE, buz çözme ısı hatası,

η, sıcaklık probu iyileştirme hatası olup her iki değer de mach sayısına göre değer almaktadır.

Şekil 2.2 ve Şekil 2.3'de DHE ve iyileştirme hatasının mach sayısına göre değişimleri verilmiştir [1].



Şekil 2.2 DHE'nin mach sayısına göre değişimi



Şekil 2.3 Sıcaklık probu iyileştirme hatasının mach sayısına göre değişimi

2.1.2.8. Serbest/statik hava sıcaklık

Serbest/statik hava sıcaklığı (T_{FAT})

$$T_{FAT} = T_t / \left(1 + 0.2 \cdot M_n{}^2\right) \qquad (2\text{-}15)$$

eşitliğinden elde edilmektedir.

2.1.2.9. Hava yoğunluk oranı

Hava yoğunluk oranı (σ)

$$\sigma = \rho / \rho_0 = 17.3345 \cdot P_s / T_{FAT} \qquad (2\text{-}16)$$

eşitliğinden elde edilmektedir.

Burada ρ hava yoğunluğu,

ρ_0 standart deniz seviyesi hava yoğunluğudur.

2.1.2.10. Atak açısı

Hava veri bilgisayarı tarafından hesaplanan atak açısı parametreleri; görüntülenen atak açısı (αi) ve doğru atak açısı (αt) verileridir. Bu parametreleri veren eşitlikler

$$\alpha i = 3/5 \cdot \left(\alpha L + 12.5\right) - 1.5 \qquad (2\text{-}17)$$

$$\alpha t = \left(\alpha L + 12\right)/1.6 - 4 \qquad (2\text{-}18)$$

şeklinde ifade edilmektedirler [1].

2.2.Yükseklikölçer

Yükseklikölçer (radar altimetre) uçağın yer yüzeyinden yüksekliğini ölçmeye yarayan araçtır. Hava veri bilgisayarı gibi, diğer tip Yükseklikölçerler uçak çevresindeki hava ile ilgili bilgileri toplayarak uçağın deniz seviyesinden yüksekliğini ölçer iken, Yükseklikölçer yer yüzeyine bir radar sinyali gönderip geri alarak uçağın yer yüzeğinden dik olarak yüksekliğini ölçmektedir.

Hesaplanmış olan yükseklikölçer bilgisi genellikle silah salım hesaplamaları ve genel uyarı sistemlerinde kullanılmaktadır. Uçak üzerine yerleştirilmiş bulunan yükseklikölçer anten sayısı sadece bir tane olduğundan uçağın her türlü

hareketinde bu ölçüm işlemi gerçekleştirilememektedir. 45 dereceden daha büyük ve –45 dereceden daha küçük yunuslama (pitch) açılarında ve 65 dereceden daha büyük yalpa (roll) ve –65 dereceden daha küçük yalpa açılarında yükseklikölçer anteni yer yüzeyine sinyal göndermeye çalıştığında sinyal yer yüzeyine ulaşamayacağından geri gelmesi mümkün olmayacaktır. Dolayısı ile hesaplama gerçekleştirilemeyecektir.

Yükseklikölçer ünitesinin aşağıda belirtildiği şekilde 4 çalışma modu bulunmaktadır.

- Kapalı mod
- Bekleme modu
- Normal mod
- Test modu

Yükseklikölçer ünitesine güç verildiğinde ünite, ilk olarak hazırlanma moduna girmektedir. hazırlanma işlemi esnasında yükseklikölçer başlangıç testini gerçekleştirmektedir. Başlangıç testi, ünitenin genel olarak faal olup olmadığını test etmektedir. Bu işlem bittiğinde ünitede bir sorun yok ise sistem otomatik olarak bekleme moduna geçmektedir. Bekleme modu, sistemin çalışmaya hazır halde beklemede olduğu moddur. Bu modda iken yükseklikölçer frekans yayınımı yapar ancak yansıyan frekansı geri almadığı için yükseklik bilgisi üretilemez. Dolayısı ile uçağın yerden yükseklik bilgisi bu modda iken elde edilememektedir.

Yükseklikölçer normal çalışma moduna geçirildiğinde, hazırlanma işlemini tamamlamışsa, fonksiyonel olarak normal çalışma moduna geçmektedir. Aksi takdirde sistem hazırlanma işleminin bitmesini beklemektedir. Bu modda yükseklikölçer frekans yayınımı ve alımı yaparak giden ve gelen sinyal arasındaki zaman farkından yola çıkarak yüksekliği hesaplar ve bu bilgi Mux–Bus veri iletim yolu vasıtasıyla ilgili sistemlere gönderilir.

Kullanıcı tarafından yükseklikölçer çalışma güvenirliği test edilmek istenirse, test işlemi kullanıcı tarafından bir buton vasıtasıyla aktif edilmektedir. Test modunda iken, yükseklikölçer yükseklik bilgisi olarak sürekli sabit ve bilinen bir yükseklik (örneğin 1000ft) üretir. Bu bilgi çeşitli göstergelerde bulunan

yükseklikölçer ile ilgili kısımda gösterilmektedir. Eğer gösterge üzerinde yukarıda bahsedilen değer görülüyor ise ünitenin doğru çalıştığı anlaşılmaktadır.

2.3.Taktik Hava Seyrüseferi

Seyrüsefer, bir yerden başka bir yere giderken yapılan yer değiştirme esnasındaki hız ve pozisyon bilgilerinin bilinmesi işlemidir. 3-boyutlu uzayda, pozisyonun ve hızın 3-boyutlu bileşenlerinden oluşan 6 bileşenli durum vektörü, aracın yaptığı tüm hareketleri matematiksel olarak ifade etmektedir. Hesaplanmış olan seyrüsefer bilgileri, uçak sistemlerinde mevcut olan hemen hemen bütün sistemlerin işleyişi için gerekmektedir.

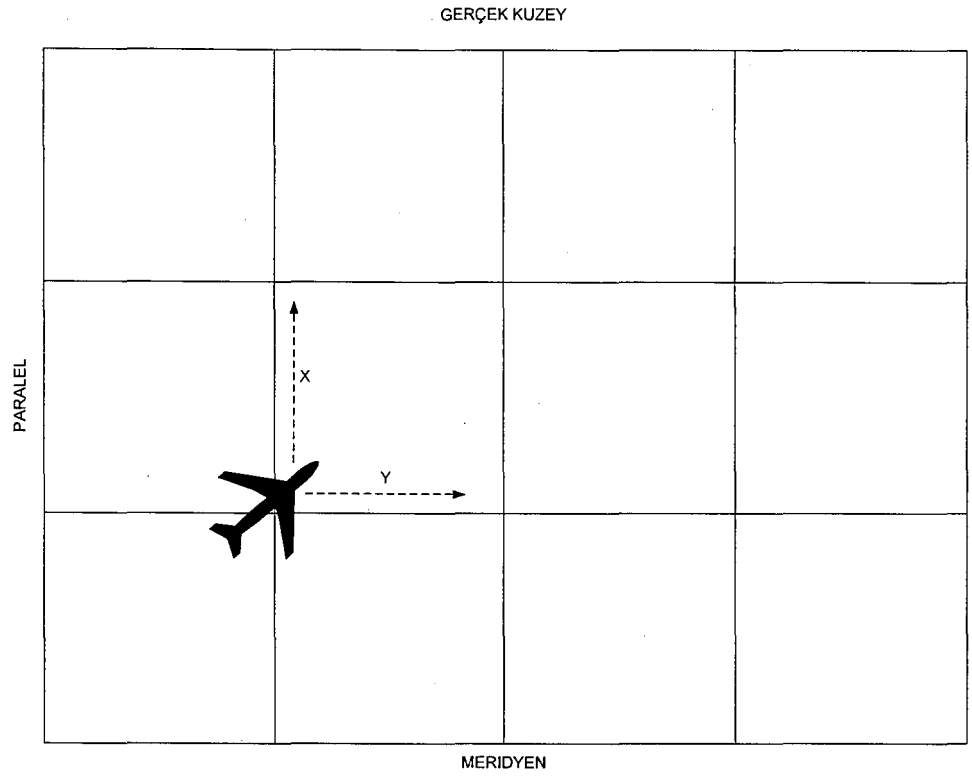
Hava seyrüseferinde dünya yüzeyinin üzerinde bir taşıma gerçekleştiğinden, seyrüsefer durum vektörlerinin dünya yüzeyine olan bağıl bileşenleri kullanılmaktadır.

Seyrüsefer, pozisyonlama (positioning) ve kendinden hesaplama (dead reckoning) olarak iki ana kategoriye ayrılabilir. Pozisyonlama sistemleri, aracın önceki pozisyon ve hız bilgisine ihtiyaç duymadan, yerde, uyduda veya başka bir araçtaki alıcı/verici ünitelerden aradaki bir haberleşme bağlantısı vasıtasıyla pozisyon bilgilerinin elde etmektedir. Örneğin, sürekli dalga üreten radyo istasyonlarının (TACAN istasyonu gibi) yaydığı manyetik alanlardan faydalanarak seyrüsefer bilgileri elde edilebilmektedir.

2.3.1.Seyrüsefer Koordinatları

Bir uçağın gerçek kuzey (true north) denilen referans yöne doğru yaptığı mesafeye paralel (latitude, x-ekseni), gerçek kuzey ile 90° yaptığı mesafeye meridyen (longitude, y-ekseni) adı verilmektedir [3]. Uçağın düşeye doğru yaptığı mesafe ise z-ekseni veya yükseklik (altitude) olarak adlandırılır[3].

Şekil 2.4'de paralel, meridyen ve gerçek kuzeyin uçuş esnasındaki bir uçağa göre pozisyonları görülmektedir.



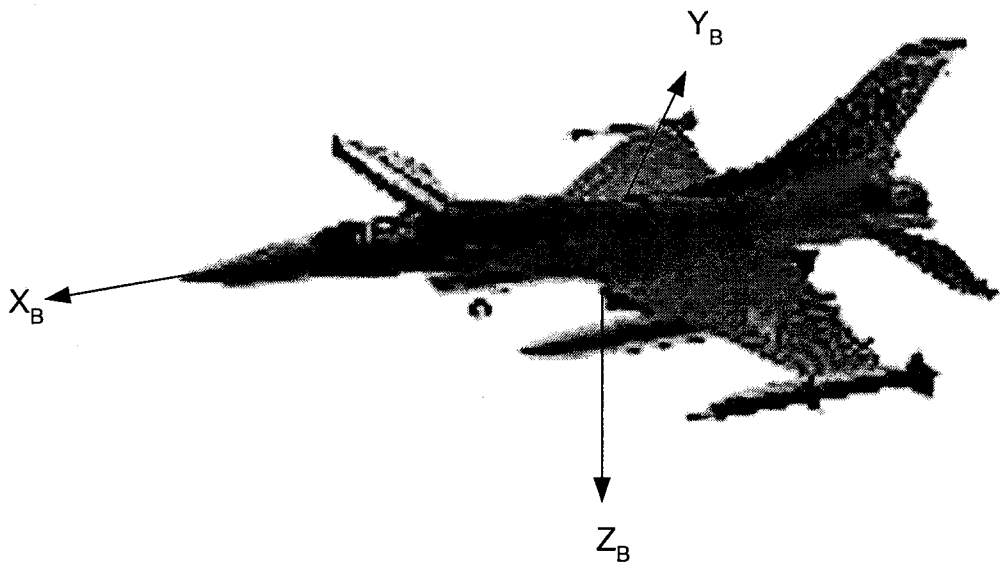
Şekil 2.4 Paralel Meridyen Eksenleri

Hava seyrüsefer ölçümlerinde seyrüsefer bilgileri elde edilmesi gereken aracın dünyaya göre yapmış olduğu yer değiştirmesinin de hesaplanması gerekmektedir. Uçağın hız ve pozisyon verileri uçağın gövdesine göre ölçülmektedir. Ancak bunun dünyaya göre bağıl vektörleri hesaplanarak dünyaya göre yapılan yer değiştirme hesaplanır.

Uçak sisteminin seyrüsefer eksenlerini bulabilmek için dünya yüzeyinde sabit bir referans noktası düşünülür. Bu sabit noktaya IR noktası denir ve dünyada sabitlenmiş bir ortagonal eksen sistemi olarak tanımlanır[4]. Bu noktanın eksenleri X_R, Y_R, Z_R olarak düşünüldüğünde, X_R ve Y_R iki boyutta koordinatlar, Z_R dünyanın merkezine doğru olan noktadır.

Seyrüsefer için tanımlanan bir diğer noktanın da uçak üzerinde olduğu varsayılır[4]. Uçak üzerinde tanımlanan bu noktaya uçak gövde eksenleri adı verilir. Gövde eksenleri uçakla birlikte hareket eder ve dönerler. Gövde eksenlerinin orijin noktaları uçağın ağırlık merkezindedir. Gövde eksenleri

(X_B, Y_B, Z_B) olarak tanımlanırsa uçak gövde eksenlerinin gösterimi Şekil 2.5'deki gibi olmaktadır [4].



Şekil 2.5 Uçak Gövde Eksenleri

Uçağın yer değiştirme ve açısal hareketleri sonucunda sürekli değişen gövde eksenleri, IR noktası orijininin, uçağın gövde eksenleri orijinine dönüştürülmesi ile elde edilirler. Bu dönüşüme uçağın oryantasyonu denir [4].

Uçağın oryantasyonu üç açı ile gösterilebilir. Bu açılar, baş açısı (heading, ψ), yunuslama (θ) ve yalpa (φ) açılarıdır. Baş açısı Z_B, yunuslama açısı Y_B ve yalpa X_B eksenine göre dönüştür. Bu üç açı kullanılarak INS Reference (IR) noktası uçak gövde eksenlerine göre tanımlanabilir. İşte elde edilen bu dönüşüme, dönüştürülmüş IR ekseni denir ve (X_T, Y_T, Z_T) ile gösterilir[4]. Dönüştürülmüş IR eksenlerinin Z_T ekseni etrafındaki ψ derece dönüşü ile elde edilen eksene sistem-1 (X_1, Y_1, Z_1), sistem-1 ekseninin Y_1 ekseni etrafındaki θ derece dönüşünü sistem-2 (X_2, Y_2, Z_2) ve sistem-2 nin X_2 ,ekseni etrafındaki φ derece dönüşünü de sistem-3 (X_3, Y_3, Z_3) olarak tanımlanabilir [4].

Dönüştürülmüş IR koordinatlarındaki herhangi bir nokta $P(P_1, P_2, P_3)$ olduğunda bu noktanın gönderimi şu şekildedir[4].

$$P_1 = \psi_{1t} \cdot P_t \tag{2-19-a}$$

$$P_2 = \theta_{12} \cdot P_1 \tag{2-19-b}$$

$$P_3 = \phi_{23} \cdot P_2 \tag{2-19-c}$$

Burada

$$\psi_{1t} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2-20}$$

$$\theta_{21} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2-21}$$

$$\phi_{32} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{2-22}$$

$\psi_{1t}, \theta_{21}, \phi_{32}$ matrisleri orthonormal dönüşüm matrisleri olarak adlandırılır [4]. Buna göre dönüştürülmüş IR eksenleri ile uçak gövde eksenleri arasındaki ilişki;

$$P_3 = \phi_{32} \cdot \theta_{21} \cdot \psi_{1t} \cdot P_t \tag{2-23-a}$$

$$P_t = \psi_{1t}^{\ T} \cdot \theta_{21}^{\ T} \cdot \phi_{32}^{\ T} \cdot P_3 \tag{2-23-b}$$

olarak bulunur. Bu matrisler yerine konulduğunda;

$$P_t = \begin{bmatrix} S\psi \cdot C\theta & S\psi \cdot S\theta \cdot S\phi + C\psi \cdot C\phi & S\psi \cdot S\theta \cdot C\phi - C\psi \cdot S\phi \\ C\psi \cdot C\theta & C\psi \cdot S\theta \cdot S\phi - S\psi \cdot C\phi & C\psi \cdot S\theta \cdot C\phi + S\psi \cdot S\phi \\ S\theta & -C\theta \cdot S\phi & -C\theta \cdot C\phi \end{bmatrix} \cdot P_3 \tag{2-23-c}$$

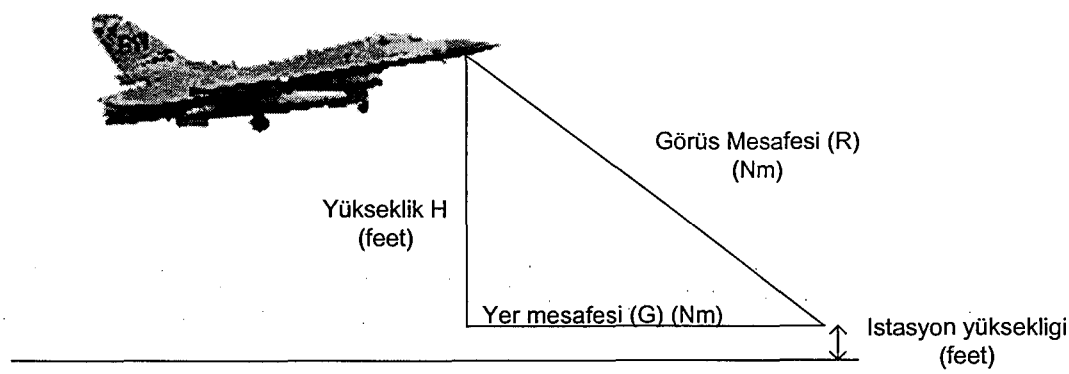
eşitliği elde edilir. (2-23-c) eşitliğindeki matrise, kosinüs dönüşüm matrisi denir. Kosinüs dönüşüm matrisi, IR noktası ile uçak gövde eksenleri arasındaki ilişkiyi veren matristir. Bu matris uçak seyrüsefer hesaplamalarında ivme ve hız vektörlerinin IR noktası – uçak gövdesi dönüşümlerinde kullanılmaktadır [2, 4, 5].

IR noktası, uçak seyrüsefer bilgilerini üreten ataletsel seyrüsefer sisteminin uçak yerde iken ayarlanması ile tespit edilen pozisyondur. Ataletsel seyrüsefer sisteminde üretilen uçak gövdesinin seyrüsefer verileri, elektronik devreleri vasıtasıyla (2-23-c) eşitliğindeki kosinüs dönüşüm matrisi kullanılarak, IR noktasına göre hesaplanarak aviyonik sisteme gönderilir[4].

Ataletsel seyrüsefer sistemleri ile birlikte, aviyonik sistemlerde seyrüsefer ihtiyaçlarını karşılamak amacıyla geliştirilmiş olan en iyi seyrüsefer çözümü yer istasyonlarını kullanarak seyrüsefer bilgileri elde etme yöntemidir. Bu metoda radyo-seyrüsefer sistemleri adı verilmektedir[6, 7].

Radyo seyrüseferinde, dünya üzerinde belirlenmiş bölgelere yerleştirilmiş olan modüle edilmiş sürekli dalga yayınlanan radyo istasyonları kullanılır. Hava taşıtları, bu referans istasyonların yayınladıkları belirli band aralıklarındaki sinyalleri alarak dünya üzerindeki pozisyonlarını belirlerler.

Radyo seyrüseferi yaparak seyrüsefer bilgileri üreten sistemler, yön bulma (direction finders DME) sistemi ve Taktik Hava Seyrüsefer (TACAN – Tactical Air Navigation) sistemidir.



Şekil 2.6 TACAN istasyonuna göre mesafe bilgisi

Seyrüsefer için gerekli olan görüş mesafesi (slant range) ile yer mesafesi arasındaki fark, uçak istasyondan çok yüksek veya istasyona çok yakın olmadığı sürece çok küçüktür. Hava aracının TACAN yer istasyonuna olan görüş mesafesi yer mesafesi ve yükseklik arasındaki ilişki dünyanın eğimi ihmal edildiğinde şu şekildedir [6]:

$$R^2 = G^2 + (H/6080)^2 \qquad (2\text{-}24)$$

Bu ilişki 30000 feet yükseklik ve 35 Nm mesafede %1 hata vermektedir [7].

Buna göre mesafe (R);

$$R = (T - 50)/12{,}359 \qquad (2\text{-}25)$$

eşitliği ile hesaplanmaktadır [7].

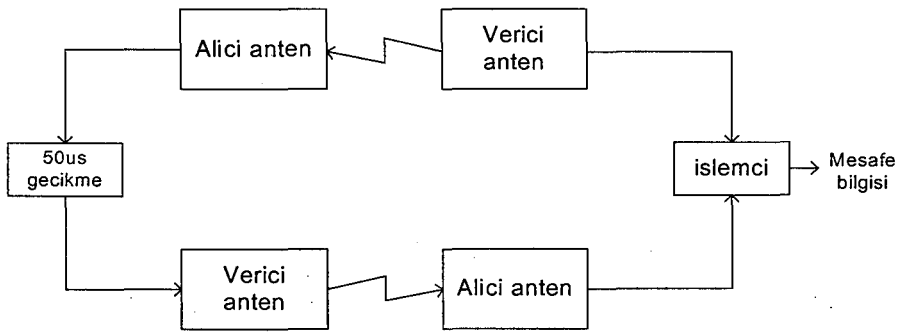
Burada R, Nm cinsinden görüş mesafesi,

T, μs cinsinden sinyal verip alma esnasında geçen zaman,

50 μs sabit yer istasyonu geciktirme süresi,

12.359 sabiti ise 1Nm mesafe için enerjinin yayınlanıp geri alınabilmesi için gerekli μs cinsinden zamandır.

Ölçülen mesafe yer istasyonuna göre belirlenmiştir. Yer istasyonunun pozisyonu sabit ve uçak tarafından bilindiğine göre, uçak dünya üzerindeki pozisyonunu bulabilmektedir. Zaman ve mesafe bilgileri kullanılarak hız bilgisi de hesaplanır. Hesaplanan hız, uçağın yer istasyonuna doğru olan hızı veya yer istasyonundan uzaklaşırken olan hızı şeklinde bulunabilir. Yer istasyonuna göre mesafe elde etme sistemi Şekil 2.7' de gösterilmiştir.



Şekil 2.7 Yer istasyonuna göre mesafe bilgisi elde etme

Dünyanın eğimi göz önüne alındığında radyo-seyrüsefer sistemleri için maksimum istasyon görüş mesafesi R.

$$R = 1.2\left((h_i)^{1/2} + (h_u)^{1/2}\right) \qquad (2\text{-}26)$$

eşitliği ile elde edilmektedir [7].

Burada R, Nm.

h_i istasyonun deniz seviyesinden yüksekliği,

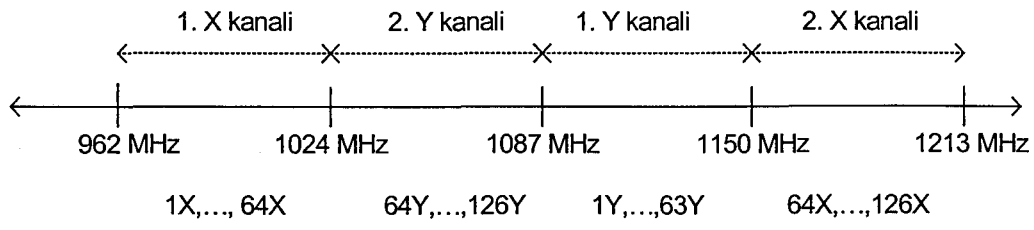
h_u uçağın deniz seviyesinden yüksekliğidir.

Uçağın yer istasyonu ile haberleşebilmesi için R mesafesini aşmaması gerekir.

Taktik Hava Seyrüseferi, askeri mesafe ölçüm ve yön bulma sistemidir ve 960-1215 MHz band aralığında çalışır. Mesafe bilgisi ile birlikte konum açısı (bearing) bilgisi de sağlar. Herhangi bir görüş hattının coğrafi alanında 136 Taktik Hava Seyrüsefer yer istasyonu bulunabilir ve her yer istasyonu 100'den fazla uçağa seyrüsefer bilgisi sunabilir [8] .

Bir TACAN yer istasyonunun 100'den fazla uçağa aynı anda seyrüsefer bilgisi sunabilmesi için her uçakla farklı frekans aralıklarında çalışması gerekir. Bu sebeple 960 MHz ile 1215 MHz arasındaki taktik hava seyrüsefer çalışma frekans bandı 1 MHz aralıklarla 126 adet X kanalı ve 126 Y kanalı olmak üzere toplam 252 kanala ayrılmıştır [7]. Böylece taktik hava seyrüsefer kanalları 1X, 1Y,.....,126X, 126Y olarak numaralandırılmışlardır. Band aralığında kanal yerleşimi Şekil 2.8'deki gibi yapılmıştır [7] . Taktik hava seyrüsefer kanalları ise Uçakta bulunan taktik hava seyrüsefer alıcı / verici ünitesi tarafından yer istasyonundan seyrüsefer bilgisi almak için frekans yayınlama işlemi (sorgulama) sadece Y kanallarında yapılır. Bu sorgulama sinyallerine, TACAN yer istasyonlarının cevabı hem X hemde Y kanallarında olabilir. Yani taktik hava seyrüseferi için 126 sorgulama kanalı, 252 cevap kanalı vardır[7].

Çizelge 2.1'deki gibidir.



Şekil 2.8 TACAN frekans band aralığı yerleşimi

Uçakta bulunan taktik hava seyrüsefer alıcı / verici ünitesi tarafından yer istasyonundan seyrüsefer bilgisi almak için frekans yayınlama işlemi (sorgulama) sadece Y kanallarında yapılır. Bu sorgulama sinyallerine, TACAN yer istasyonlarının cevabı hem X hemde Y kanallarında olabilir. Yani taktik hava seyrüseferi için 126 sorgulama kanalı, 252 cevap kanalı vardır[7].

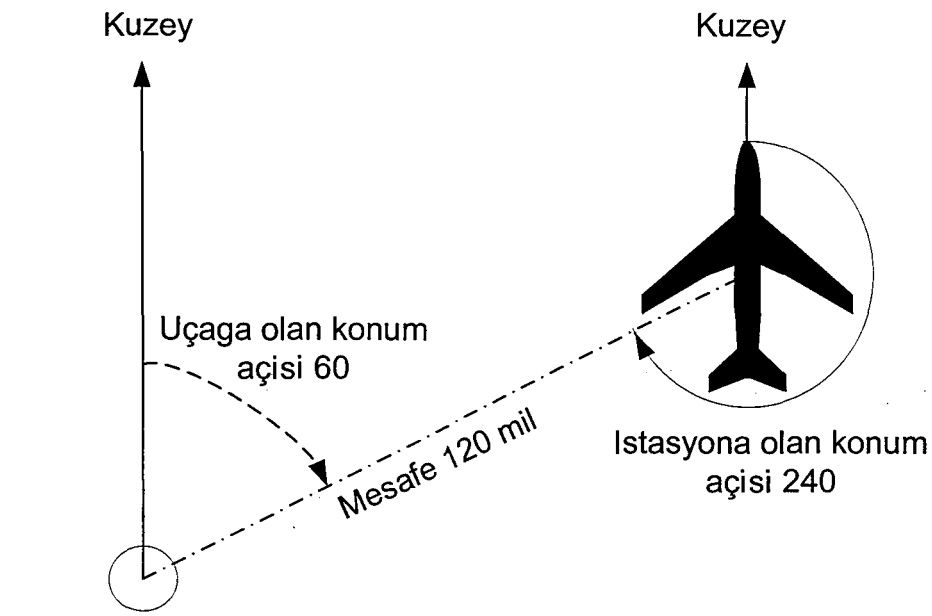
Çizelge 2.1 TACAN kanalları

Kanal	Frekans (MHz)
1X	962
2X	963
....	....
63X	1024
64Y	1025
.....	.....
126Y	1087
1Y	1088
.....	.....
63Y	1150
64X	1151
.....	.....
126X	1213

Taktik hava seyrüsefer sistemi, uçağa pozisyon, hız bilgileri ile beraber yön bilgisi de sağlamaktadır. taktik hava seyrüsefer sistemince ölçülen yön bilgisi konum açısı olarak adlandırılmaktadır. Konum açısı uçağın kuzeyi ile istasyonun yaptığı açıdır. Yani konum açısı bilgisi sürekli olarak kuzey yönünü referans alarak uçağa yön bilgisi sağlar. Şekil 2.10'da konum açısı görülmektedir[3].

Konum açısı bilgisi uçağın taktik hava seyrüsefer alıcısına, TACAN yer istasyonu tarafından gönderilir.

Uçakta bulunan taktik hava seyrüsefer ünitesi için TACAN yer istasyonu sorgulama frekansı band genişliği 1025 – 1150 MHz arasındaki Y kanallarıdır. Seçilen Y kanalına bağlı olarak 126 mümkün sorgulama frekansı vardır. X ve Y kanal sinyalleri pulse yapıdadır. Sinyallerin zamanlaması kanallarına frekansına göre değişmektedir. TACAN yer istasyonu tarafından gönderilen cevap sinyalleri ise 252 mümkün frekans kullanabilirler [7].



Şekil 2.10 Konum açısı (bearing)

Taktik hava seyrüsefer sisteminin Y kanal frekanslarında yayınlanıp, X ve Y kanal frekanslarında geri aldığı sinyaller arasındaki fark bulunarak, uçağın istasyona ulaşma zamanı ve istasyona olan mesafe hesaplanır. Mesafenin zamanla türevi alındığında uçağın TACAN yer istasyonuna göre hızı elde edilir. Bu işlemler uçakta bulunan taktik hava seyrüsefer ünitesinin dijital devrelerinde gerçekleştirilir. Böylece taktik hava seyrüsefer ünitesi yön, pozisyon, hız ve zaman seyrüsefer bilgilerini aviyonik sisteme sunmuş olur[7].

TACAN yer istasyonlarınca yayınlanan bir diğer bilgide kimlik (ident) bilgisidir. Her TACAN istasyonunun kendine has bir kimlik sinyali vardır. Bu

sinyal 1350 KHz frekansında bir ses sinyalidir ve her TACAN istasyonu için farklı tondadır. İstasyondan gönderilen frekansın içinde bulunan kimlik ses sinyali, uçağın taktik hava seyrüsefer alıcı ünitesi tarafından alınarak uçağın dahili haberleşme sistemi vasıtasıyla pilotların kulaklıklarına gönderilir. Kimlik sinyali ancak (2-21) eşitliğinde verilen maksimum TACAN istasyonu mesafesi sınırları içinde alınabilir. Böylece pilot bir TACAN istasyonunun yayınım mesafesi içerisine girdiğini anlar [7].

2.3.2.Taktik hava seyrüsefer fonksiyonel operasyonları

Taktik hava seyrüsefer sistemi, yerdeki TACAN istasyonlarından veya referans TACAN istasyon uçağından sinyal sorgulaması şeklinde haberleşme ile seyrüsefer bilgileri sağlayan bir sistemdir. Taktik hava seyrüseferinin REC, T/R, A/A REC ve A/A, T/R olmak üzere 4 fonksiyonel operasyonu vardır [7].

REC modunda taktik hava seyrüsefer ünitesi sadece seçilen yer istasyonundan aldığı konum açısı sinyalini üretir. A/A REC modunda, referans TACAN istasyon uçağı tarafından yayınlanan konum açısı sinyalini alarak konum açısı bilgisi üretir. T/R modunda, yer istasyonu tarafından gönderilen sinyalleri alarak konum açısı ve mesafe bilgilerini üretir. A/A T/R modunda referans TACAN istasyon uçağı tarafından yayınlanan seyrüsefer sinyallerini alarak konum açısı ve mesafe bilgilerini üretir[3].

2.3.3.Taktik hava seyrüsefer çalışma durumları

Taktik hava seyrüsefer'ın kontrolü, TACAN kontrol panel adı verilen, bir panel vasıtasıyla manuel olarak yapılır. TACAN kontrol panel ile taktik hava seyrüsefer ünitesi, 'çoklu hat '' denilen özel bir seri haberleşme hattından iletişim sağlanmaktadır. Taktik hava seyrüsefer ünitesi de ürettiği seyrüsefer bilgilerini synchro hatlar vasıtasıyla aviyonik sisteme aktarır.

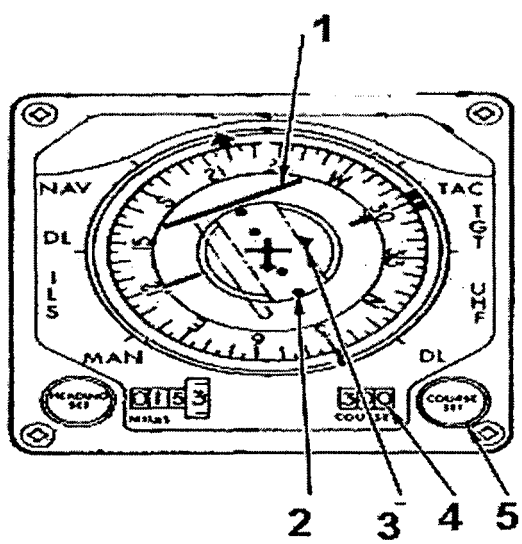
2.3.4.TACAN istasyonu ile ilgili gösterimler

TACAN istasyonu ile iletişim sağlandığında uçak üzerinde Yatay Durum Göstergesi (HSI - Horizontal Situation Display) üzerinde istasyonun yeri ile ilgili bazı bilgiler görüntülenmektedir (Şekil 2.12). Bunlar taktik hava seyrüsefer ünitesinin hesapladığı konum açısı, mesafe bilgisi ve bunlardan başka To/From

bayrağı ile istasyona yaklaşma yönünden sapma (Course Deviation) bilgileridir. To/From bayrağı (Şekil 2.12-No3) ve İstasyona yaklaşma yönünden sapma hesaplamaları yatay durum göstergesi tarafından gerçekleştirilmektedir. Yatay durum göstergesi analog bir gösterge olup bunun yerine günümüz teknolojilerinde aynı görevi Aviyonik bilgisayarının sürdüğü dijital göstergeler de gerçekleştirmektedir. Bu durumda bu hesaplamaları Aviyonik ünite gerçekleştirmektedir.

Yaklaşma yönü, manuel olarak yatay durum göstergesi ünitesi üzerinden seçilmektedir (Şekil 2.12-No4,5). Yaklaşma yönü, pilotun seçilen TACAN istayonuna, kuzeye göre hangi açıdan yaklaşmayı istediğini göstermektedir. Eğer seçilen yaklaşma yönü açısı ile istasyona yaklaşılamaz ise seçilen yaklaşma yönünden sapma açısı da bu gösterge üzerinde gösterilmektedir. (Şekil 2.12-No1). Aşağıda şekli verilen yatay durum göstergesinde 2 numara ile gösterilen noktalar yaklaşma yönünden sapma açısını göstermektedir. Her bir nokta 5 dereceyi göstermektedir, ancak ünitenin hassasiyetine göre bu noktaların anlamları üniteden üniteye değişim göstermektedir..

Yatay durum göstergesinde üzerindek bir diğer imleç To/From bayrağı seçilen istasyona doğru (To) ya da seçilen istasyondan uzaklaşmayı (From) göstermektedir.



Şekil 2.12 Yatay durum göstergesi - HSI

2.4.Dost/Düşman Tanıma Sistemi:

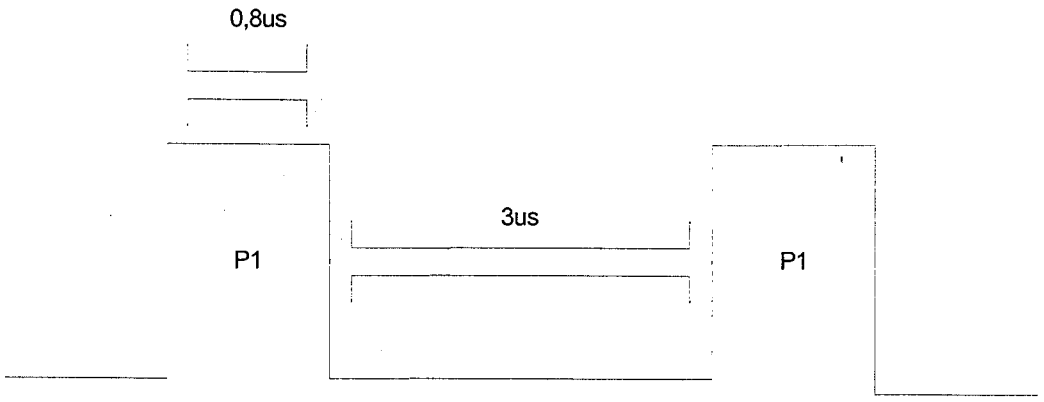
Denizde, havada ve karada seçilebilir kimlik tanıma ve sorgulaması yapan sistemlere Dost/düşman tanıma (IFF - Identification Friend or Foe) adı verilir. Dost/düşman tanıma sistemi vasıtasıyla gönderilen kodlanmış radyo sinyalleri, tanımlama istenilen diğer dost/düşman tanıma sistemlerince alınır. Bu sistemlerden gelen cevaba göre, dost ve düşman tanımlaması yapılır. Dost/düşman tanıma sistemi askeri hava araçlarında kullanılmaktadır. Karadan dost/düşman tanıma istasyonları vasıtasıyla havadaki uçan araçlar sorgulanabildiği gibi, havadan havaya da sorgulama ve diğer hava araçlarından gelen kimlik tanımlama isteği sinyallerine karşı cevaplama da mümkündür. Sorgulama (interrogation) için çalışma frekansı 1030 MHz, cevaplama (reply) için çalışma frekansı ise 1090MHz'dir [9].

Dost/düşman tanıma sorgulama ve cevaplamaları için kullanılan özel sinyaller vardır. Bunlar dost/düşman tanıma haberleşme modları olarak bilinirler. Bu modlar ve sinyal yapıları askeri ve sivil havacılıkta standart olup şu şekilde isimlendirilmişlerdir [9].

- Mod 1
- Mod 2
- Mod 3/A
- Mod C
- Mod 4

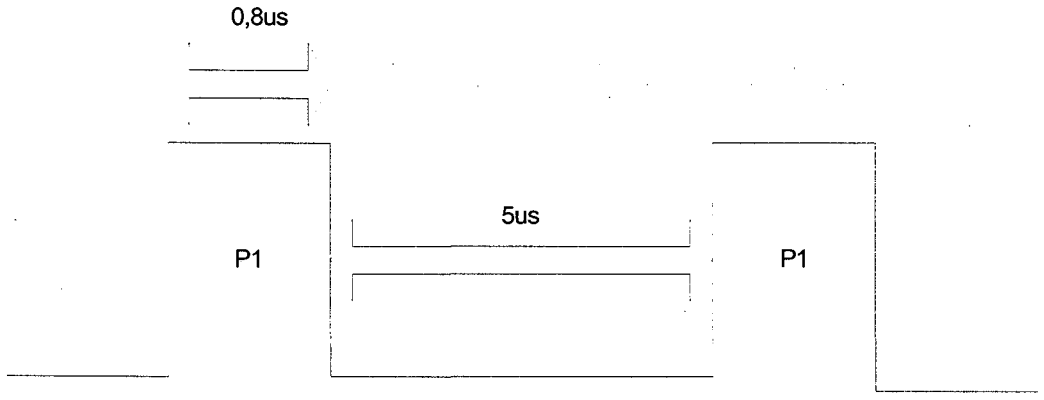
Dost/düşman tanıma sisteminin bu modları kullanarak sorgulama ve cevaplama özelliğine SIF denir.

Mod 1; hava trafik, yer istasyonu yardımı ve genel tanımlama amaçları için kullanılan askeri bir Dost/düşman tanıma haberleşme sinyalidir. Hava aracının kimlik bilgisini içerir. Uçuş esnasında duruma göre kullanıcı pilot tarafından değiştirilebilir. Sorgulama sırasında Mod 1'in sinyal yapısı 3 μs'lik puls dizini şeklindedir [9]. Bu Şekil 2.13' de görülmektedir.



Şekil 2.13 Dost/düşman tanıma Mod 1 sorgulama sinyali yapısı

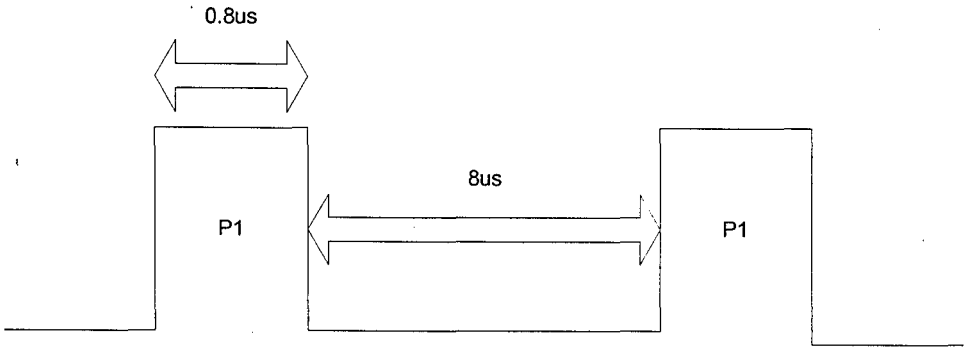
Mod 2, askeri gizli kimlik bilgilerini içeren dost/düşman tanıma haberleşme sinyalidir. Hava aracının bir ünitesi durumunda olan dost/düşman tanıma sistemi üzerindeki kodlama anahtarı yardımıyla uçuştan önce Mod 2 kod seçimi yapılır. Uçuş esnasında kullanıcının bu modu değiştirme şansı yoktur. Sorgulama sırasında puls dizini 5μs aralıklıdır (Şekil 2.12) [9].



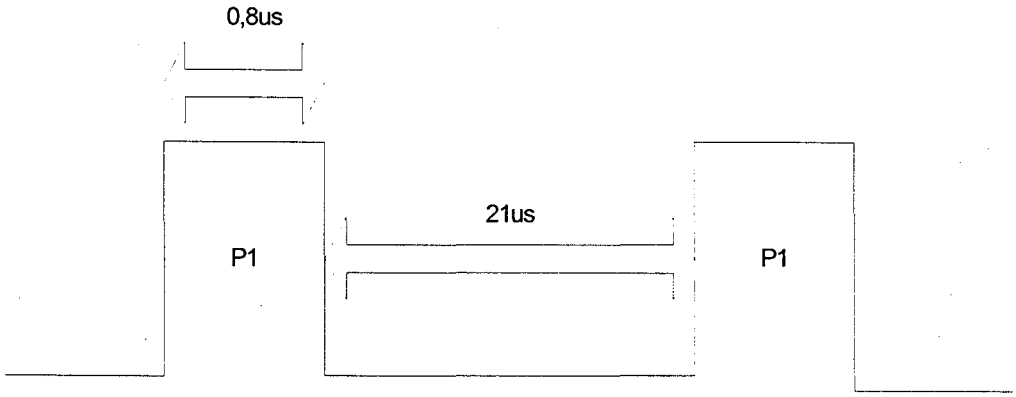
Şekil 2.14 Dost/düşman tanıma Mod 2 sorgulama sinyali yapısı

Mod 3/A, hava alanlarında bulunan trafik kontrol sistemlerine yardımcı olmak için kullanılan bir dost/düşman tanıma haberleşme sinyalidir. Askeri Mod 3/A, sivil hava araçlarında kullanılan Mod 3 ile uyumludur [9]. Sorgulama esnasında puls dizini 8μs aralıklıdır (Şekil 2.13) [9].

Mod C, sivil ve askeri hava araçlarında beraber kullanılan uçuş yüksekliği bilgisini yer istasyonuna bildiren dost/düşman tanıma haberleşme sinyalidir. Sorgulama esnasında puls dizini 21μs aralıklıdır(Şekil 2.14) [9].



Şekil 2.15 Dost/düşman tanıma Mod 3/A sorgulama sinyali yapısı



Şekil 2.16 Dost/düşman tanıma Mod C sorgulama sinyali yapısı

Mod 4, sadece özel amaçlar için kullanılan özel şifreli tanımlama için genelde savaş durumunda kullanılan dost/düşman tanıma haberleşme sinyalidir. Şifreleme işlemi Mod 4 Bilgisayarı adı verilen özel bir ünite ile yapılır. Bu sistem uçuş esnasında dost/düşman tanıma sistemine entegre edilerek kullanılır. Her ülkenin kullandığı Mod 4 şifresi farklıdır. Mod 4'ün kullanımı sırasında hem sorgulayan hemde cevaplayan dost/düşman tanıma sisteminde bulunan sinyali şifreler, cevaplayıcı dost/düşman tanıma sisteminde bulunan Mod 4 Bilgisayarı ise bu şifreyi çözerek dost/düşman tanıma ünitesine gönderir[9].

2.4.1.SIF dizini

Dünyada üretilen tüm dost/düşman tanıma sistemlerinde standart olarak kullanılan SIF özelliği bir puls trenidir. A, B, C ve D puls grupları olarak

kodlanmış bu puls trenine göre,, Mod 1, 2, 3/A ve C sinyal yapıları belirlenir. Mod 1, Mod 2, Mod 3/A ve Mod C sorgulama sinyallerinden herhangi birisini alan dost/düşman tanıma sistemi buna göre uygun SIF puls trenini hazırlar ve sorgulamanın cevabı olarak gönderir. Mod 4 için SIF kullanılmaz. Mod 4 haberleşmesi çok gizli ve özel bir sinyal haberleşmesi olduğundan, SIF işlemi Mod 4 Computer tarafından gerçekleştirilir[9].

Mod 1, 2, 3/A ve C için puls tanımlamalarıÇizelge 2.2'deki gibidir [10]

Çizelge 2.3 Dost/düşman tanıma modlarının SIF trenine göre tanımlamaları

Mod 1	A1 A2 A4 B1 B2
Mod 2 ve Mod 3/A	A1 A2 A4 B1 B2 B4 C1 C2 C4 D1 D2 D4
Mod C	A1 A2 A4 B1 B2 B4 C1 C2 C4 D1 D4

Mod tanımlamalarında kullanılan pulslar ikilik kodlama yöntemi ile şifrelenirler. Buna göre A grubu için A1 LSB, A4 MSB'dir. Bu diğer gruplar içinde geçerlidir. Örneğin ; Mod 1 kod 71, Mod 2 kod 7763, Mod 3/A 7122 ve Mod c 1124 cevaplamaları için puls dizinleri çizelge 2.3'deki gibidir[10].

Çizelge 2.4 Dost/düşman tanıma modlarının SIF trenine göre örnek kodları

Mod 1	A1 A2 A4 B1 B2
Kod 71	1 1 1 1 0
Mod 2	A1 A2 A4 B1 B2 B4 C1 C2 C4 D1 D2 D4
Kod 7763	1 1 1 1 1 1 0 1 1 1 1 0
Mod 3/A	A1 A2 A4 B1 B2 B4 C1 C2 C4 D1 D2 D4
Kod 7122	1 1 1 1 0 0 0 1 0 0 1 0
Mod C	A1 A2 A4 B1 B2 B4 C1 C2 C4 D1 D4
Kod 1124	1 0 0 1 0 0 0 1 0 0 1

Hava aracında herhangi acil bir durum olduğunda, dost/düşman tanıma sistemi bu acil durumu bildirme ve yardım isteme için de kullanılır. Bunun için askeri ve sivil standartlarda kabul edilmiş modlar vardır. Acil durum esnasında, kullanıcı tarafından dost/düşman tanıma sistemi, acil (EMERGENCY) moduna alındığında; otomatik olarak Mod 1 kod 70, Mod 2 kod 7777 ve Mod 3/A kod 7700 bilgilerini gönderir ve acil durumu diğer hava aracı ve yer istasyonlarına bildirir [9].

2.4.2.Dost/düşman tanıma sisteminin operasyonel modları

Genel olarak dost/düşman tanıma sisteminde şu modlar bulunmaktadır.

- Kapalı Mod (OFF): Dost/düşman tanıma sistemine enerji verilmez ve IFF çalışmaz

- Bekleme Modu (STBY): Dost/düşman tanıma sistemiçalışır durumdadır ancak gelen sorgulamalara cevap verilmez.

- Düşük Seviye Modu (LOW): Dost/düşman tanıma sistemi sorgulama ve cevaplama yapmaktadır ancak sinyal hassasiyeti düşüktür.

- Normal Seviye Modu (NORM): Dost/düşman tanıma sistemi tam kapasite ile sorgulama ve cevaplama yapmaktadır.

- Acil Durum Modu(EMRG): dost/düşman tanıma sistemiin acil durumda çalışmasını sağlar. IFF EMRG konumuna alındığında Mod 1 kod 70, Mod 2 kod 7777 ve mod 3/A kod 7700 olarak gönderilir. Mod C ve Mod 4 ise normal sinyal haberleşmesine devam eder [10]

Hava aracına dışarıdan bir dost/düşman tanıma sorgulaması geldiği anda kullanıcı pilot gözle ve sesle ikaz edilir. Dost/düşman tanıma sistemi bir 28VDC/OPEN kesikli sinyalle IFF lambasını yakar. IFF sesli uyarı sinyali de aktif edilerek, "head set" adı verilen pilot kulaklığına "IFF" sesli uyarı sinyalinin gelmesini sağlar. Böylece pilot, bir dost/düşman tanıma sorgulamasına maruz kalındığını anlar. Bir Mod 4 sorgulaması geldiği anda ise dost/düşman tanıma sistemi tarafından yine bir 28VDC/OPEN kesikli sinyalle direk olarak Mod 4 Uyarı lambası yanar ve pilot ikaz edilir.

Dost/düşman tanıma cevaplaması kokpitte bulunan IFF kontrol paneli üzerinden aktif edilmiş modlarla gerçekleştirilir. Aktif olmayan mod cevaplama dizininde bulunmaz.

3. HAVACILIKTA SİMÜLASYON SİSTEMİ

Bu bölümde, aviyonik sistemlerin uçak sistemleri ile olan ilişkisi ve buna göre modelde sahip olacağı yeri kavrayabilmek için genel olarak uçak sistemleri ve simulatör sistemlerinin yapısından bahsedilmiştir.

Bu sayede gerçekleştirilen modellerin ihtiyacı olan girdi bilgilerini elde edebilecekleri diğer sistemler daha kolay kavranabilecektir.

3.1.Simulasyon Sistemleri

İnsanoğlu en iyi deneyerek öğrenir. Bir Çin atasözü en iyi öğrenmenin deneyimsel olduğunu şu şekilde ifade eder; "Söylersen unuturum, gösterirsen hatırlarım, yaptırırsan öğrenirim". Yürümek, bisiklete binmek ve piyano çalmak deneyerek öğrenilmektedir. Deneyimsel öğrenmenin kökeninde, bir hedefe ulaşmak için deneme yapılması, denemenin sonuçlarının alınması ve sonuçlarının kişi tarafından yorumlanarak bir sonraki denemenin daha başarılı yapılması yatmaktadır. Bisiklete binmenin nasıl öğrenildiği düşünüldüğünde deneyimsel öğrenmenin en güçlü öğrenme yöntemi olduğu çok açıktır. Bununla birlikte, deneyimsel öğrenmenin gerçekleşebilmesi için deneyden alınan geri dönüşün hızlı ve belirli olması gerekmektedir [11].

Oysa ki gerçek yaşamda karmaşık sistemlerle karşı karşıyayız ve pek az sistem bize - bisiklet kullanmayı öğrenmede olduğunun aksine- anında ve belirli geri dönüşler verebilir. Örneğin bir pilot yapmış olduğu uçuşta, gerçekleştirmiş olduğu manevranın ya da mühimmat atışının doğruluğunu ancak uçuş sonrası toplantısı sırasında öğrenebilecek; atmış olduğu mühimmatın hedefi vurup vurmadığını atış sahasından gelecek değerlendirme raporundan sonra öğrenebilecektir. Bu durumda manevrayı ya da atışı yaptığı sırada içinde bulunduğu durumu değerlendirmesi atladığı ya da hatırlayamadığı bazı olaylarla beraber değerlendirmesine neden olacaktır. Öyleyse sonuçlarını direkt olarak gözlemleyemediğimiz durumlarda deneysel öğrenme bize nasıl yardımcı olabilir? [11].

Simülasyonlar, normal şartlarda gözlemleyemediğimiz olayları keşfetmemizi ve deneyerek öğrenmemizi sağlayan uygulamalar olarak karşımıza

çıkmaktadır. Uzun vadede gerçek deneyimler ve gerçek yanılmalar ile ulaşılabilecek tecrübenin, bilgisayar tarafından canlandırılan tamamen güvenli bir ortamda kazanılmasını mümkün kılmaktadırlar. Böylelikle uzun sürelerle kazanılabilecek spesifik bir uzmanlığın daha kısa sürelerde kazanılması mümkün olmaktadır[11].

Uçak gibi hava araçlarının kullanımının öğrenilmesi pahalı, riskli ve de süreci uzun işlemlerdir. İnsanoğlu kullandıkça pratiklik kazanır. Bir pilot uçuş okulundan başlayarak emekli oluncaya kadar uçuşlarının çoğunu eğitim amacıyla gerçekleştirmektedir. Uçak sistemlerinin karmaşıklığı ve de kullanılmadıkça pratikliklerin unutulduğu düşünülecek olursa uçuş eğitimlerinin iş hayatı boyunca yapılmasının ne kadar normal olduğu ortaya çıkmaktadır. Uçuş sırasında karşılaşılacak riskli durumlar, zamanın en kıymetli ve önemli olduğu anlardır. Bu sebeple uçak içinde yapılacak her türlü işlemin pratik ve seri olarak gerçekleştirilmesi gerekmektedir ki güvenli bir uçuş gerçekleştirilebilsin.

Gerçek şartlarda gerçekleştirilen uçuş eğitimlerinin en önemli eksikliği acil durum eğitimlerinin ve her türlü silah eğitiminin tam anlamıyla yapamamalarıdır. Bunun sebebi bu tür eğitimlerin hayati risk taşımasıdır. Daha önce karşılaşmadığı bir durum ile öğrenci pilot ilk kez eğitiminde karşılaştığında bunu hayatı ile ödeme riski oldukça fazladır.

Bir havayolu firmasının ya da hava kuvvetlerinin eğitimler için yapmış olduğu masraflar oldukça fazladır. Ülke savunmasında önemli yer tutan hava kuvvetlerinin pilotlarının mevcut kıtalarında hem uçuş eğitimi almaları hem de silah talimlerini gerçekleştirmeleri gerekmektedir. Her bir pilotun günde en az bir kere uçuş ve silah eğitimi yaptığı düşünülürse ülke ekonomisi için bunun ne kadar büyük bir yük olduğu ortadadır.

İşte yukarıda sayılan tüm sebepler göstermektedir ki bir uçağın simulatörünün gerçekleştirilmesi her açıdan oldukça önemlidir. Bu sayede öğrenci pilot uçuşunu seri olarak gerçekleştirmeyi, havada karşılaşabileceği inişte motor arızası, açılmayan iniş takımı, jeneratörün kaybı gibi acil durumlarda nasıl davranması gerektiğini, her türlü silah atışını hem hayati hem da mali olarak daha ucuz bir şekilde öğrenebilecektir .

Simulasyonlar gerçek bir sistemi yazılım yolu ile taklit ederek, sistem ile etkileşim kuran kişiye anında geri dönüşler verirler. Bununla birlikte gerçek bir sistemi taklit etmek, sistemin analitik olarak çözümlenmesini ve bilgisayar ortamında yeniden programlanmasını gerektirir. Basit sistemlerde oldukça basit olan bu işlem karmaşık yapıların canlandırılmasında zor bir işlem olabilir.

Simulasyonlar, hızlı ve kalıcı öğrenmeye imkan sağlamaları sayesinde, gün geçtikçe daha fazla kullanılan güçlü ve yaygın uygulamalar olma yolundadırlar.

Simulatör sistemleri yazılım ve donanım olarak ikiye ayrılabilir. Donanım kısmında, kokpit, girdi/çıktı kontrolleri, monitörler, görsel sistem, ses sistemi, yük kontrol sistemi ve hareket sistemi bulunmaktadır. Simulatör yazılımları da uçağın simulasyonunu gerçekleştiren, eğitmen ile simulatör arasında arayüzü oluşturan bilgisayar programlar bütünüdür [12].

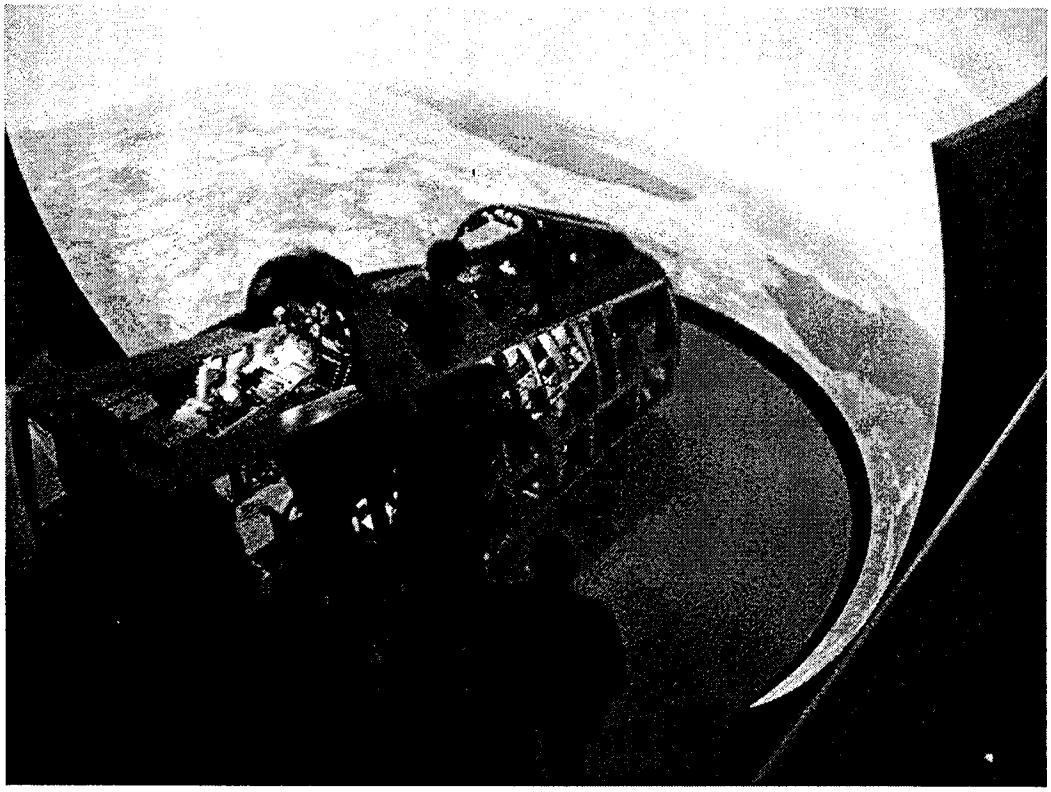
Simulatörün şekil ve performansı seçilen simulatör tipine göre çeşitlilik gösterebilmektedir. Ancak ana olarak bir simulatör aşağıdaki ekipmanlara sahiptir [12]:

- Kokpit (öğrenci konsolu)
- Bilgisayar Sistemleri
- Girdi/çıktı Sistemleri
- Eğitmen/Operatör Konsolu
- Görsel Sistem

Aşağıdaki sistemler ise simulatör sistemlerinde ihtiyaca göre opsiyonel olarak bulunmaktadır:

- Ses Sistemi
- Hareket Sistemi
- Yük Kontrol Sistemi

Genel olarak bir simulatörün görünümü Şekil 3.1'deki gibidir.

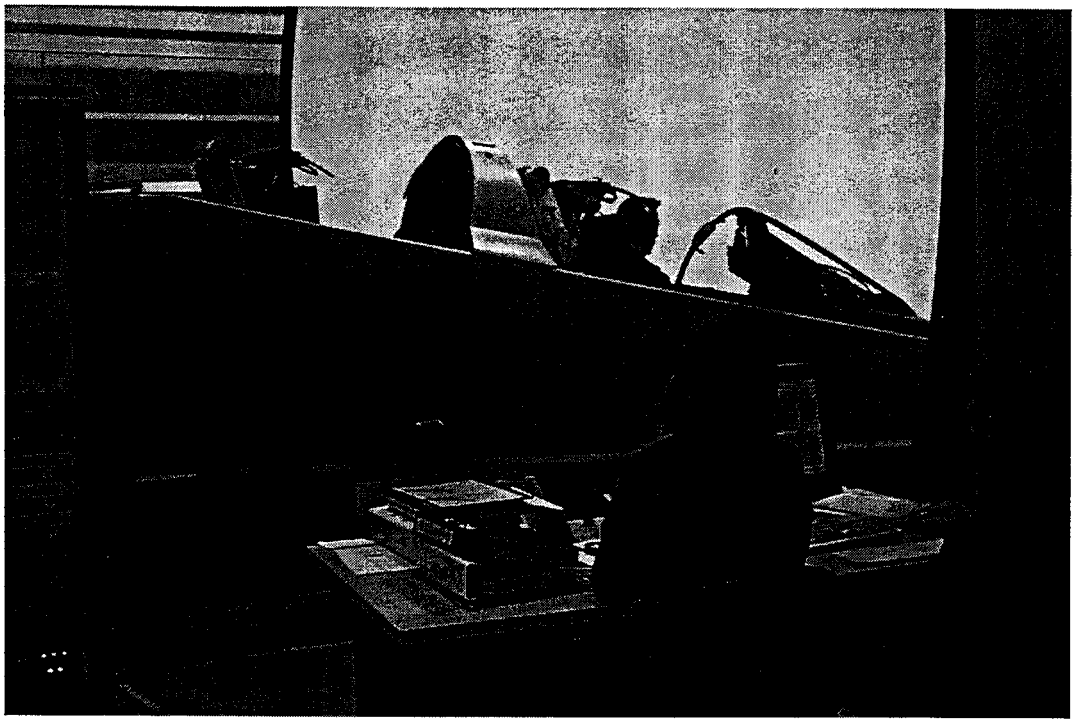


Şekil 3.1 Uçuş Eğitim Simulatörü

3.1.1.Kokpit (öğrenci konsolu)

Simule edilen uçak geniş gövdesi olan bir taşıma uçağı, helikopter ya da savaş uçağı olabilir, her ne olursa simulatörün gerçekçi olması, öncelikle kokpitin görünüşünün simulatörü yapılan araç ile aynı olması gerekmektedir. Genellikle uçağın orijinal kokpiti alınarak kullanılmakla birlikte, daha ekonomik ve kokpitte kullanılacak simule edilmiş ya da ek olarak kokpite takılacak parçaların da yerleşimi düşünülerek, kokpitin orjinaline yakın fiberglas'dan üretilmiş bir kokpit daha etkin olarak kullanılmaktadır [12].

Kokpit içine yerleştirilecek her bir enstrumanın orjinal mi yoksa simule mi olacağı tamamen parçanın üreticisinden bulunabilmesine ve maliyetine göre şekillenmektedir. Enstrumanın simule olması durumunda önemli olan husus, gerçek kokpittekinden farkının olmaması gerektiğidir (Şekil 3.2) [12].



Şekil 3.2 Kokpit

3.1.2.Bilgisayar sistemleri

Bilgisayarlar simulatör sistemlerin tam anlamıyla kalbidir. Simulatörde gerçekleşen en basit olayın bile gerçekleşmesinden sorumludur. Bilgisayar sistemleri, kokpit ile gerçek zamanlı arayüzler ile haberleşmektedir. Yukarıda sayılmış, simulatör sistemine ait ekipmanlar arasındaki iletişim ve bilgi alışverişini sağlayarak gerçekleştirilen simulatör sisteminin işleyişini gerçekleştirmektedir [12].

3.1.3.Girdi/çıktı sistemleri

Girdi/çıktı sistemleri kokpit ile diğer sistemler arasındaki iletişimi sağlamak için bulunmaktadır. Kokpit içerisinden öğrencinin yapmış olduğu herhangi bir değişiklik Girdi/çıktı sistemi sayesinde ilgili bilgisayar ya da sistemlere iletilir. İletilmiş olan değişiklik, ilgili sistemde işlendikten sonra yine Girdi/çıktı sistemi ile kokpitte ya da görselde, bir tepki olarak öğrencinin karşısına çıkar.

Kokpit ve simulasyon bilgisayarları arasında bilgi aktarımını gerçekleştiren alt sistemler genellikle girdi-çıktı kontrolörleri (I/O Controller) olarak

adlandırılırlar. Girdi ile kokpitten bilgisayarlara gelen sinyaller, çıktı ile de bilgisayarlardaki prosesler sonucunda ortaya çıkan ve genellikle kokpit'e gönderilen sinyaller anlatılmaktadır.

Bu konuda en yaygın olarak kullanılan sistem VME (Versa Module Eurocard) bus sistemidir. Motorola, phillips, Thompson ve Mostek tarafından 1981'de ortaya çıkarılmış açık uçlu (open ended) esnek bir sistemdir. 21 adet kartın takılabileceği bir kafes şeklinde tasarlanmış olup ilk yuvada kafesteki tüm kartların yönetimini sağlayan ana kart bulunmaktadır. Bu ana kart hesaplamaları yapan bir bilgisayar mahiyetindedir ve tüm bellek de bu kart üzerinde bulunmaktadır [12].

Girdi/çıktı kontrolör konfigürasyonu ana olarak dijital girdi, dijital çıktı, analog girdi, analog çıktı kartlarından oluşmaktadır [12].

Dijital girdiler genellikle radyo kanal seçme anahtarı, devre kesici anahtarlar gibi kokpitteki seçme anahtarlarından oluşmaktadır. Dijital çıktılar, ikaz ve uyarı lambaları gibi genellikle kokpit içerisinde bulunan bazı indikatörleri yakamak için kullanılmaktadır. Analog girdiler, levye ve dümen gibi sürekli değişen voltaj değerlerinde olan kontrolleri okumak için kullanılmaktadır. Analog çıktılar ise altimetre gibi sürekli voltaj değişimi olan indikatörlerin sürülmesinde kullanılmaktadır. Synchro çıktılar da yakıt akış göstergesi gibi faz kayması olabilen değişken indikatörler için kullanılmaktadır.

Bu sistemin içerisinde bulunan güç üreteçi AC 'yi DC'ye çevirmektedir. Genel olarak kartların ihtiyacı olan 28V, 12V, ve 24V üretilmesini sağlamaktadır.

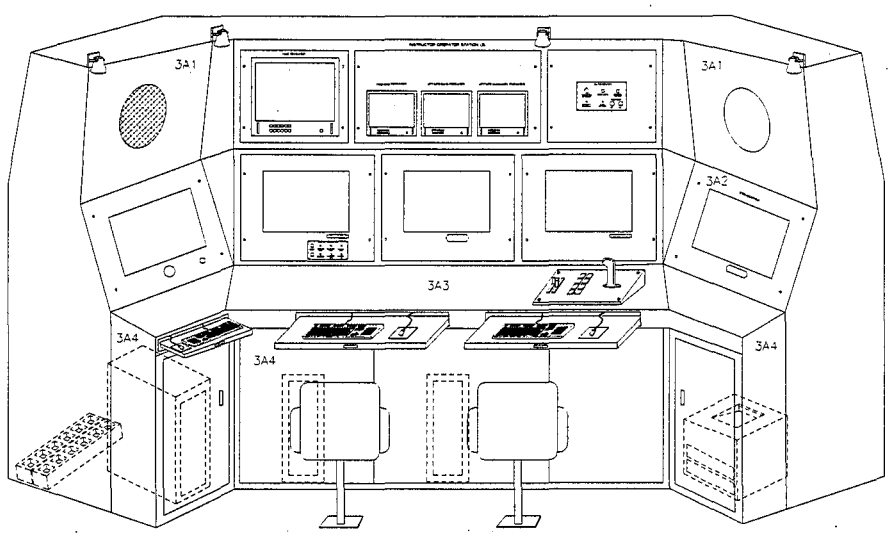
3.1.4.Eğitmen/operatör konsolu

Eğitmen konsolu, simule edilmiş kokpitin bir uzantısı olarak eğitimin kontrolünü ve analizini yapmak amacıyla kullanılmaktadır.

Eğitmen konsolu, öğrenci pilotun uçuş sırasında dış dünyadan alması gereken tepki ve etkileri düzenlemektedir. Eğitmen kişi konsolundan aşağıdaki operasyonları sağlayabilmektedir (Şekil 3.3) [12]:

- Öğrencinin eğitimi için senaryo hazırlama,
- Öğrencinin eğitimi sırasında eğitime ve senaryoya yön verebilme,

- Öğrencinin eğitimi sırasında eğitim uçuşunu durdurup, istediği noktadan tekrar eğitime devam edebilme,
- Öğrencinin yapmış olduğu hareketleri değerlendirerek dinamik olarak eğitime yön verebilmekte,
- Öğrenciye yapmış olduğu hareketleri gösterip doğru bir geridönüş verebilmek için eğitimi kayıt edip daha sonra üzerinden tartışabilmekte,
- Gerçek hayatta karşılaşabileceği acil durumları yine bu konsoldan aktif ederek öğrencinin acil durumlara karşı eğitilmesini sağlamakta,
- Kullanılmakta olan tüm silahlarla ilgili olarak atış teknikleri ve savunma tekniklerini öğretebilmektedir.



Şekil 3.3 Kokpit

3.1.5.Görsel sistem

Görsel Sistem, gerçek dünyayı simule etmek amacıyla kullanılmaktadır. Kullanılan görsel sistem, tamamen kullanıcının isteğine göre 40 dereceye 30 derece boyutunda bir ekran da olabilir, 360 derece görüş alanı olan bir kubbe de

olabilir. Bu tamamıyla simulatör tasarlanırken verilen karara göre değişmektedir [12].

Bir görsel sistem, bilgisayar görüntü üretecinden (CIG – Computer Image Generator) ve görüntüleme sisteminden (Display System) meydana gelmektedir. Görüntü üreteci ana bilgisayara bağlı basit bir grafik kartı da olabilmekte, isteğe bağlı olarak ayrı olarak yapılandırılmış paralel işlem yapabilen gerçek görüntüyü elde edebilecek kapasitedeki yüksek kalitede bir bilgisayar sistemi de olabilmektedir. Görüntüleme sistemi ise tek kanalı ya da 360 derecelik görüş alanı sağlayacak nitelikteki 3 kanallı bir sistem de olabilir. Eski CIG sistemleri tek hattan 1 kanal sürebilirken, bugün tek hattan 8 kanala kadar sürüş yapabilme kabiliyetindedir. Grafik hatlarının 210 derecelik bir görüş alanı oluşturması 3 ayrı hattan 3 kanalın sürülmesi ile de gerçekleştirilebildiği gibi tek hattan 3 kanalın sürülmesiyle de gerçekleştirilebilir. İkisinin arasındaki fark 1 hattın oluşturabildiği poligon sayısının farklılığı olur. Oluşturulan poligon sayısındaki artış meydana gelen görüntünün kalitesini arttırmaktadır. Bu sebeple 1 hattan 1 kanal sürülmesi, görüntünün kalitesi için önemlidir [12].

IG terimi genellikle görüntüyü oluşturan donanımı anlatmak için kullanılmaktadır. Normal bir bilgisayar, görüntü üretme işlemini, kompleks bir senaryoyu oluşturacak hesaplamaları gerçekleştiremeyecek kadar yavaştır. Bu sebeple, çoğu sistem görüntü üretebilmek için, yüksek hızlı, dijital grafik işleyebilen bilgisayarlarla sistemlerini desteklemektedir. Dijital grafik işlemcileri, grafik motoru olarak adlandırılmakta ve grafik hatlarının ana bileşenini oluşturmaktadır [12].

Görüntü üreteci Image generator (IG), CIG görsel sisteminin işlerinin büyük bir bölümünü gerçekleştirmektedir. Bu sebeple görsel sistemi genelde IG sistemi olarak adlandırılmaktadır. Ancak IG sistemi kadar görüntüleme ve veri tabanı sistemi de görsel sistemin önemli parçalarıdır [12].

Uçuş simulatörlerinde, pilotun istediği oranda kokpitin ve dış dünyanın gerçeğine uyması beklenmektedir. Simulatör kokpitinin gerçek kokpitle uyumunun yanısıra, görüntüleme sisteminde düşünüleni elde etmek pilotlar için oldukça önemlidir. Çünkü uçuş sırasında dış dünyada karşılaşılan her bir obje ya

da olay uçuş sırasında hissetikleri görüş açısı, parlaklık, çözünürlük simulasyonunda da gerçekleştirilirse o zaman simulatörün başarıya ulaştığı düşünülmektedir. Pilotlar için gerçek dünyada hissettiklerini hissediyor olmak inancı oldukça önemlidir [12].

Tek kanal görüntü elde etmek, 3 boyutlu sürekli hareket eden, değişen dış dünyanın verilen görüş yerine göre iki boyuta indirilmesi anlamına gelmektedir. Bu da verilen bir yerin fotoğrafını çekmekle aynı anlama gelmektedir. 3 boyutlu ortamda haritalanmış objelerin de 2 boyuta aktarılırken sapma olmadan, uzaydaki doğru yerine konuşlanmış olarak gösteriyor olması gerekmektedir ki, simulatör uçuşlarında pilota yanlış hissiyat verilmesin. Günümüz uçak teknolojilerinde kullanılmakta olan Head Up Display (HUD), Helmet gibi aviyonik üniteler üzerinde bulunan sembolojilerin de yeryüzü şekilleri ile uyumlu olması gerektiği de düşünülürse, görüntü sistemi üzerindeki her bir objenin doğru konuşlandırılmasının ne kadar önemli olduğu ortaya çıkmaktadır [12].

Görüntüleme sistemlerinde 5 önemli parametre şunlardır. Görüş alanı, ambiyans parlaklık, renklendirme, çözünürlük ve görüntünün çıktı formatı. Görüş alanı, kullanılacak olan optiğin büyüklüğünü belirlediği için görüntüleme sistemindeki yeri önemlidir. Ambians parlaklığı, lens, ayna, ekran gibi optik bileşenlerin optik verimliliklerini ayarlamak için ve görüntü sisteminin çıktı parlaklıklarını belirlemek için kullanılmaktadır. Mesela gündüzün simulasyonunda gerekli olan parlaklık gece ve alacakaranlık için gerekli olandan oldukça fazladır. Parlaklık, kullanılan görüş alanı büyüdükçe azalması gereken bir parametredir. Renklendirme parametresi simulatör sisteminde kullanılacak renklendirme tipini belirler. Monokrom, sınırlı renk ya da tüm renkler gibi opsiyonlardan birinin seçimi ile görsel sistemin kalitesi değişebilmektedir. Çözünürlük ise görsel sistemin kalitesinde etkili olan bir diğer parametredir. Bunun iyi seçimi ile görsel sistemde objelerin belirginlik seviyesi değişmektedir. Çözünürlüğün değeri genellikle kullanılan perde sisteminin boyutuna göre değişmektedir. Son parametre olan görüntünün çıktı formatı IG ile görüntüleme sistemi arasındaki arayüz formatıdır. RS170, RS343 gibi bir çok arayüz formatı bulunmaktadır. Arayüz formatları ana olarak iki kategoride toplanabilir. Bunlar Raster ve Calligraphic formatlarıdır. En yaygın olarak kullanılan format,

televizyon sistemlerinde de kullanılan raster formatıdır. Bu formatta görüntü bilgisi IG'den paralel hatlar serisi olarak piksel piksel soldan sağa şeklinde okunur. Bir görüntü karesi soldan sağa ve yukarıdan aşağı şeklinde satır satır ekran üzerine yapılandırılır. Yapay ve dikey sapmaların da IG tarafından bildirilmesiyle hangi zamanda hangi karenin işleneceği belirlenmiş olur [12].

Calligraphic formatta, her bir piksel X ve Y olarak tanımlanmıştır. Görüntü bu sayede, raster formata göre daha çabuk ve basit bir şekilde çizilebilmektedir. Calligraphic görüntüleyiciler, raster görüntüleyicilere göre daha karmaşık ve pahalı sistemlerdir. Aynı zamanda raster bir cihazdan alınan verimden daha iyisi elde edilemez. Calligraphic format günümüz teknolojilerinde stroke format olarak da adlandırılmaktadır [12].

Görüntüleme sistemlerinde bir diğer önemli nokta ise görüntüleme parametrelerinin hesaplanmasıdır. Ayarlanmasına ihtiyaç duyulan parametreler şunlardır :

- görüş alanı (FOV – Field of View)
- Parlaklık
- Kontrast
- Çözünürlük
- Renk
- Renk Uyumu
- Geometri
- Collimation mesafesi

Renk uyumu genellikle direk görüntüleme sistemi üzerinde, diğer altı öğe simulatörün göz nokta dizaynına (design eye point – optimize olarak en iyi optik performansın alındığı, pilotun görsel ve aviyonik sisteme oturuşunun en uygun olduğu nokta) göre ayarlanmaktadır [12].

3.1.6.Ses sistemi

Ses sistemi uçuş sırasında duyulan her türlü sesi simule eden sistemdir. Ses sistemi pilotun hem kulaklıklardan duyduğu uyarı ve ikazları, hem de uçağın çalışması sırasında çıkan sesleri simule etmek için kullanılmaktadır.

Ses sistemleri, bilgisayar tabanlı, piyasada bulunabilen, stereo amplifier sistemlerdir. Sınırsız sayıda ses üretilebilmekte olup, dijitize edilmiş sesler yazılım ile bir araya getirilip, birden fazla ses aynı anda da çıkarılabilmektedir.

3.1.7.Hareket sistemi

Simulatörü yapılmakta olan uçağın kokpiti hareket eden bir platform üzerine yerleştirilmiş olabilir. Hareket sisteminin hareket sınırı, yunuslama ve yalpa hareketlerini sağlayan 2 derece özgürlük veren sistemlerden, uçağın etrafında ve eksenlerinde hareket özgürlüğü olarak adlandırılan 6 dereceye kadar serbestlik (6 degree of freedom)veren sistemlere kadar değişebilmektedir. 6 derece serbestlik olarak bahsedilen hareketler, yunuslama (pitch), yalpa (roll), sapma (yaw), sallanma (sway), ilerleme (surge) ve yükselme (heave) olarak adlandırılmaktadır. Yunuslama uçak burnunun yapmış olduğu yukarı ve aşağı hareketleri, yalpa uçağın burun ekseni etrafındaki dönüş hareketleri, sapma burunun sağa sola hareketleri, yükselme uçağın tümüyle yukarı aşağı hareketi, ilerleme uçağın ileri geri hareketi ve sallanma uçağın sağa sola kayış hareketini ifade etmektedir [12].

Pilotun simulatörde yapmış olduğu uçuş sırasında yunuslama hareketini hissedebilmesi önemlidir. Çünkü pilot kazanmış olduğu yunuslama hissiyatıyla uçağın davranış, hız ve yüksekliğini kontrol etmektedir. Yalpa hissiyatını kazanma da pilotlar için önemlidir çünkü, kazanmış olduğu yalpa hissiyatıyla uçağın, baş kontrolünü, uçuş yolu kontrolünü ve endirek de olsa yüksekliğini kontrol etmektedir.

Bahsedilen iki ana hareketin yanında yunuslama hareketini belirginleştirmek için heave hareketinin, yalpa hareketinin belirginleştirmek için de sapma hareketi ve sallanma hareketinin de sistem de mevcut olması eğitimin daha etkin verilebilmesi için önemlidir.

Hareket sisteminin bir amacı da, pilotun uçuş sırasında karşılaşmış olduğu G yükünü gerçek şartlardaki gibi üzerinde hissedebilmesidir. Hareket sistemi haricinde bu hissiyatı verebilecek g sandalyesi (g-Seat) ve G kıyafeti'de (g-Suit) simulatörlerde kullanılmaktadır. Hareket sistemleri sadece g hissiyatını verebilmek için değil ayrıca havada karşılaşılan türbülans gibi durumların hissini verebilmek için de kullanılmaktadır [12].

Hareket sistemi, her bir hareket özgürlüğünün verebilmesi için her bir özgürlüğe ayrı ayrı hareket kontrolörü, girdi/çıktı sistemi, hidrolik pompa, hidrolik soğutma sistemi ve hidrolik hareket ettirici bulundurmaktadır. Hareket geribeslemeleri sayesinde de durum bilgileri elde edilmekte ve bu sayede kontrolsüz hareketler engellenmektedir [12].

Simulatör sistemlerinde 6 derece serbestlik her zaman hareket sistemlerince verilmemektedir. 6 derece serbestlik hissi kokpitin sabit durması ve görselin hareketiyle de pilota hissettirilebilmektedir. Görsel sistem bu hareketleri, pilotun kokpit içinden yapmış olduğu levye, pedal hareketleri sonucunda oluşturmaktadır [12].

3.1.8.Yük kontrol sistemi

Bu sistem pilot kontrol sistemine bağlı olup pilotun uçuş sırasında kontrol mekanizmaları üzerinde hissettiği yükleri hissettirmek üzere tasarlanmıştır. Genel olarak yay sistemlerinden oluşmaktadır ancak günümüzde mikroişlemcilerin kontrol ettiği elektriksel ya da hidrolik sistemler olarak çalışmaktadırlar [12].

6 derece serbestliği sağlayan kokpit içindeki tüm sistemlerin yük kontrolü bu sistem sayesinde sağlanmaktadır. Bu sayede pilot 6 derecede de hareket hissiyatını simulatör sistemlerinde elde etmektedir [12].

3.2.Uçak Sistemleri

Simulatörü gerçekleştirilecek sistemin iyi anlaşılması modellemenin başarılı bir şekilde gerçekleşmesi açısından oldukça önemlidir. Her bir sistem girdi olarak çeşitli verilere ihtiyaç duymaktadır. Eğer her model aynı veriyi kendi başına elde etmeye çalışırsa, her biri aynı veriyi çok küçük de olsa farklı değerlerde elde edebilir. Bu gibi küçük farklar bile tüm sistemin birarada çalışması sırasında

anlam verilemeyecek büyük uyumsuzluklara neden olabilmektedir. Bu sebeple sistemlerin simulatörde çalışma prensipleri ayrıntılı şekilde anlaşılmalı ve hangi verinin nereden geleceği tasarımı oldukça dikkatli gerçekleştirilmelidir.

Bu tezin de konusu olan aviyonik sistemi, uçak sistemlerinin alt sistemidir. Bu sistemin diğer sistemlerle etkileşiminin iyi anlaşılabilmesi için uçak sistemlerinin genel prensiplerinden bahsetmek faydalı olacaktır.

Genel olarak uçakta şu alt sistemler bulunmaktadır :

- Güç Sistemi
- Elektrik Sistemi
- Yakıt Sistemi
- Hidrolik Sistem
- İniş Takım Sistemi
- Fren Sistemi
- İklimlendirme, Basınç ve Oksijen Sistemi
- Uçuş Kontrol Sistemi
- Yangın Haber Sistemi
- Aviyonik Sistem
- Uçuş Dinamiği

3.2.1.Güç sistemi

Güç sistemi genel olarak uçak motoruna güç üretmek için kullanılmaktadır. Uçak motorları havada da çalıştırılabileceği için hem havada hem de yerde güç verme ve motorları çalıştırma olaylarının gerçekleştirilmesi bu sistem kapsamındadır. Bütün bu işlemler pnömatik hava, yakıt, ateşleme gibi çalışma şartlarının sağlanmasıyla gerçekleşmekte ve motorun operasyonu da tüm motor çalışma seviyelerine göre sağlanması gerekmektedir. Yukarıda sayılan, içinde bulunan şartlara göre motorun sağlayacağı güç ve hareket miktarı mevcut tablolardan çıkarılabilmektedir. Yakıt yakış miktarına göre kalan yakıt miktar

hesaplamaları gerçekleştirilmekte ve pnömatik motor çalıştırıcıların kontrolü ise öğretmen konsolundan gerçekleştirilmektedir [12].

3.2.2.Elektrik sistemi

Kokpitteki elektrik sistemi tasarımı uçak üzerindeki elektrik sistem tasarımına uygun olarak gerçekleştirilmelidir. Simule edilen elektrik hatları ve devre kesicilerin her biri ilgili olduğu sistem enstrümanlarının güçlerinin kontrol edilmesini sağlamaktadır. Bunun sebebi, uçuş sırasında özellikle acil durumlar için pilotların gözü kapalı bir şekilde ilgili sistemlerin devre kesicilerini kontrol edebilmelerinin gerekmesidir [12].

Uçağın ilk çalıştırılması sırasında gerekli olan harici güç kontrolünün eğitmen konsolundan gerçekleştirlmesi ile kokpit için gerekli olan AC güç sağlanmaktadır.

3.2.3.Yakıt sistemi

Uçak üzerinde dahili ve harici olmak üzere iki tip tank yakıt deposu bulunmaktadır. Dahili tanklar gövde ve kanat içi olmak üzere iki yerde bulunmaktadır. Harici tanklar ise gövde altı ve kanat uçlarını istek olursa takılmaktadır. Uçak sistemleri genelde yakıtı gövde içerisindeki tanklardan kullanmaktadır. Yakıt azaldıkça kanat içlerinden ve harici tanklardan yakıt transferi yapılmaktadır [12].

Yukarıda bahsedilen tüm yakıt işlemlerinin simulatör sistemlerinde de gerçekleştirilmesine ihtiyaç vardır. Bu nedenle eğitmen konsolundan harici yakıt tanklarının yüklenmesi ve dahili yakıt tanklarına yakıt yükleme ve acil durum koşulunun gerçekleştirilebilmesi için boşaltma işlemleri gerçekleştirilmektedir.Kokpit içerisindeki indikatörler de dahili tank yakıt miktarına göre değer göstermektedir [12].

3.2.4.Hidrolik sistemi

Hidrolik sistem, uçuş kontrol ünitelerine gerekli olan hidrolik gücü sağlamaktadır [12]. Hidrolik sistem sayesinde iniş takımları, kanat ve kanatçık, fren sistemi gibi uçuş kontrol ünitelerinin hareketi sağlanmaktadır.

3.2.5.İniş takım sistemi

İniş Takım sistemi kokpit içerisindeki iniş takım kontrol anahtar ve kolları ile iniş takım durum indikatörlerinden oluşmaktadır. İniş takımlarının konumuna göre uçaktaki bazı aviyonik sistemlerin mod ve durumları belirlenmektedir. Uçağın yerde olması durumunda kapalı olmaması gereken sistemler de bu sistemden elde ettiği girdiler ile sistemlerini aktif etmektedir [12].

3.2.6.Fren sistemi

Fren sistemi kokpit içindeki kontrolleri ve hidrolik sistemi ile beraber çalışmaktadır [12]. Havada ve yerde uçağın yavaşlaması için kullanılan bir kumandadır. Havada iken bu sistem kanat ve kanatçıkları etkiler iken yerde iniş takımlarına fren görevini yapmaktadır.

3.2.7.İklimlendirme, basınç ve oksijen sistemi

Uçuş şartlarına göre kokpit içerisinin iklimlendirilmesi ve uçak üzerindeki g etkisinden dolayı basıncın ayarlanması gerekmektedir. Bununla beraber pilotun kapalı ortamda bulunmasından ve yüksek irtifalarda bulunmasından dolayı oksijene de ihtiyacı bulunmaktadır. Simulatör sistemlerinde genellikle bu sistem tam anlamıyla gerçeklenmemektedir. Ancak bulundukları irtifaya göre bu sistemler ile ilgili indikatörler göstermeleri greken değerlere sürülmektedir. Bunun sebebi de gerçek uçuştaki kullanım alışkanlıklarının öğrenci pilota kazandırılmasıdır [12].

3.2.8.Uçuş kontrol sistemi

Kokpit içerisindeki kanatçık (aileron), irtifa dümeni (elevator), istikamet dümeni (rudder) kontrolleri uçuş modeli ile gerçekleştirilmektedir. Bunların kullanımı sırasında pilotun kazandığı hissin simulasyonu da yük kontrol sistemi ile gerçekleştirilmektedir.Yük kontrol sistemi hidrolik değişimler ve manuel olarak yapılan kontroller ile kontrol yüzeylerinin ayar (trim) fonksiyonları elektriki sinyaller vasıtasıyla gerçeklenmektedir [12].

3.2.9.Yangın haber sistemi

Yangın haber sistemi eğitmen konsolundan kontrol edilmekte ve verilen değere göre kokpit içerisindeki indikatör ve ses sistemleri etkilenmektedir [12].

3.2.10. Aviyonik sistem

Aviyonik sistem aşağıda sayılan sistemlerin simulasyonunu içermektedir.

- Dahili/eğitmen komunikasyon sistemi (ICS – Intercomm System)
- Radyo Sistemi
- Dost/düşman Tanıma Sistemi
- Taktik Hava Seyrüsefer Sistemi
- Ataletsel Seyrüsefer Sistemi
- Hava Veri Bilgisayar Sistemi
- HUD/MFD gösterge sistemi

Dahili/egitmen komunikasyon sistemi, kokpit minimum 2 kişi ise öğrenci pilotların kendi aralarında konuşmaları ve de eğitmen ile iletişimlerini sağlayan sistemdir.

Radyo sistemi eğitmen ve öğrencinin mevcut radyo frekans ve kanalları aynı ise iletişim yapmalarının sağlayan sistemdir.

Dost/düşman sistemi, haberleşilen uçağın dost/düşman ayırımını yapan sistemdir. Bu sistemde eğitmen, öğrenci pilotun kokpit içerisinden girmiş olduğu değerleri kontrol edebilmekte ve buna göre gerekli yayınımları sağlayarak, pilotun dost/düşman uçak eğitimini sağlamaktadır.

Taktik hava seyrüsefer sistemi, öğrenci pilotun kokpit içerisinden seçmiş olduğu TACAN istasyonuna olan seyrüsefer bilgilerinin HUD, HSI gibi göstergelere sürülmesini sağlayan sistemdir.

Ataletsel seyrüsefer sistemi uçağın bulunduğu pozisyon bilgilerini üreten sistemdir. Bu kapsamda seçilen bir noktaya gitmek için seyrüsefer bilgileri üretilerek göstergelere sürülmesi gibi fonksiyonlar da gerçekleştirilmektedir.

Hava veri bilgisayar sistemi uçağın çevresindeki mevcut hava bilgilerini toplayarak, uçağın irtifası, sıcaklık, hava yoğunluğu gibi bilgileri üreterek gerekli sistemlere göndermektedir.

HUD/MFD gösterge sistemi, yapılan tasarıma göre yukarıda bahsedilen aviyonik sistemlerden elde edilen bilgilerin gösterimini gerçekleştirmektedir.

3.2.11. Uçuş dinamigi

Uçuş dinamigi uçağın uçuş performansını belirleyen modeldir.Bu model, uçağın toplam ağırlığının hesabından, uçağın uçmakta olduğu hız ve yüksekliğe kadar uçuş ile ilgili her türlü bilgiyi hesaplayan modeldir. Bu modelde hesaplanan veriler şu şekilde sınıflandırılabilir[12]:

- Ağırlık ve denge
- Uçuş katsayıları
- Güç ve momentler
- Hareketin dönüsel denklemleri
- Hareketin dönüsüm denklemleri
- Atmosfer
- Yer tepkileri

Bu sınıflandırmaya göre ağırlık ve denge ile ilgili olarak uçağın toplam mühimmat ve yakıt tanklarıyla beraber toplam ağırlık ve buna göre değişen uçak ağırlık merkez hesaplamaları, ataletsel moment hesaplamaları gerçekleştirilmektedir. Uçuş katsayıları sürekli ve anlık olarak hesaplamaktadır. Bu katsayılar uçak üzerine etkiyen hız, yükseklik ve AOA gibi değerlerin hesabında kullanılacak kuvvetleri hesaplamak için gereklidirler. Güç ve moment hesaplamaları, hesaplanan uçuş katsayıları kullanılarak gerçekleştirilmekte ve hareketin büyüklük ve yön ile ilgili hesaplamalarını gerçekleştrmektedir. Hareketin dönüsel denklemleri ile uçağın yapmış olduğu açısal ivme, hız ve davranış hesaplamalarını gerçekleştirmektedir. Hareketin dönüsel denklemlerinin hesabında uçağın mevcut ağırlık ve ağırlık merkez hesapları ile uçağın üzerine etkiyen kuvvet ve momentler de kullanılmaktadır. Hareketin dönüşüm denklemleri, herhangi bir zamanda uçağın üç eksendeki doğrusal hız ve ivmelerini hesaplamaktadır. Uçak üzerinde elde edilen hız, ivme ve pozisyon bilgileri daha sonra bu bilgilerin coğrafik pozisyon bilgilerinin elde edilmesinde kullanılacaktır.

Atmosfer kısmında uçağı çevreleyen atmosferik özellikler belirlenmektedir. Buna göre çevresel basınç oranı, çevresel hava sıcaklığı, mach sayısı gibi değerler elde edilmektedir. Yer tepkileri kısmında ise uçak yerde iken iniş takımları ve uçağa uçak ilerlerken ya da dururken etkiyen güç ve moment hesaplamalarını gerçekleştirmektedir[12].

4. SİMULASYON MODELLERİNİN HAZIRLANMASI

Bu bölümde, daha önce çalışma prensiplerinden bahsedilmiş olan hava veri bilgisayarı, yükseklikölçer, taktik hava seyrüseferi ve dost düşman tanıma ünitelerinin modellenmeleri incelenmektedir.

Simulasyon modelleri, her bir simulasyon alt sisteminde gerçek şartları sağlayabilmek için ihtiyaç duyulan yazılımlardır. Bu model iki sistem arasındaki arayüzü oluşturacak seviyede de oluşturabilir, sistemin kendisini de oluşturuyor olabilir.

Daha önceden de bahsedildiği gibi, simulatörlerde kullanılması mümkün olmayan ünitelerin modellerinin hazırlanmasına ihtiyaç duyulmaktadır. Bu durumda, modelleri hazırlanacak ünitenin matematik modelleri, çalışma prensipleri çıkarılmasına ihtiyaç duyulmaktadır. Çoğu ünitenin kokpit içerisinde kontrol panelleri bulunmaktadır. Bu panellerden ünitenin çalışması, kapatılması, çalışma modu belirlenmektedir. Kokpit içinden yapılan bu tip değişiklikler ve de kokpitin çevreden alması düşünülen tepkiler ile modellere girdi sağlanmakta ve çalışması buna göre belirlenmektedir.

Tüm aviyonik ünitelerin kontrol mekanizmaları vardır. Bunlar test (BIT - Built In Test) olarak adlandırılan, ünitenin çalıştırılması sırasında, periodik olarak ya da kullanıcının isteği üzerine yapılabilen ünite fonksiyonel testleridir. Gerçekleştirilen fonksiyonel test sonucunda ünitede bir arıza bulunmuş ise, bulunan arıza kokpit içerisindeki bir göstergede rapor edilir ve ortaya çıkan arızaya göre ünite ve ünite ile alakalı sistemler gerekli tepkileri oluştururlar.

Simulatör sistemlerinde modeller ideal şartlara göre çalışmaktadır. Ancak daha önce de bahsedildiği gibi simülatörün en önemli amaçlarından biri hava şartlarında gerçeklenemeyen arızaların gerçeklenmesi olduğu için, her bir ünite modellenirken bu modellerin arızalanmaları durumunda sistemin vereceği tepkiler de düşünülmelidir. Eğitmen pilot istediği arızayı eğitmen konsolundan aktif ederek arıza durumunu gerçekleştirmektedir. Burada bahsedilen her bir ünite için test fonksiyonu ve temel olarak her ünitede bulunabilecek güç kesintisi ya da arayüz kesintisi gibi arızaların modellenmesi gösterilmiştir.

Modellerin temel olarak bulunan fonksiyonlarının yanı sıra üreticisine göre ya da sisteminde tasarlanmasına göre de ekstra fonksiyonları bulunabilmektedir. Bu tezde genel olarak her bir ünitede bulunabilecek fonksiyonlar ele alınmış ve her bir ünitenin modellenmesinde kullanılabilecek farklı arayüzlerden bahsedilmiştir.

Modellerin hepsi C++ dilinde yazılmıştır [13]. Modellerin kaynak kodları Ek-1, Ek-2 ve Ek-3'de sunulmuştur.

4.1.Hava Veri Bilgisayar Modeli

Hava veri bilgisayarı bahsedildiği gibi uçağı çevreleyen sensörlerden gelen havanın karakteristiği ile ilgili bilgileri derleyerek ilgili diğer aviyonik sistemlerin ve de pilotun uçuş boyunca ihtiyacı olan basınç ve barometrik yükseklik, statik ve toplam basınç, doğru ve ayarlanmış hız, mach sayısı, hava yoğunluğu, sıcaklık gibi atmosferik değerleri standart ya da standart olmayan hava durumlarına göre üretmektedir.

Simulatörlerin konuşlandırıldığı ortamlarda, çevre koşulları (ortam sıcaklığı, hava hızı gibi) sabit olduğundan, dış ortam sensörleri olan sıcaklık, pitot-statik ve hücum açısı sensörlerinin kullanılmasının sisteme pozitif bir katkısı yoktur ve bu nedenle simulatörlerde kullanılmazlar.

Bu dış ortam sensörleri simulatörlerde kullanılmadığı için hava veri bilgisayar modeli oluşturulurken yükseklik ve gerçek hava hızı (TAS – True Air Speed) gibi parametreler sisteme girdi olarak verilir ve diğer parametreler de bu değerler dikkate alınarak hesaplanır. Bu yüzden hava veri bilgisayar modelinde gerçeğinden farklı matematiksel yöntemler kullanılarak hava verileri elde edilmektedir.

Hava veri bilgisayar modelinin çalışması için ihtiyacı olan girdi bilgileri ve bu bilgilerin edinildiği simulatör sistemleri şunlardır;

- Uçuş modelinden elde edilen girdiler:
  - Deniz seviyesinden yükseklik (ft)
  - Gerçek hava hızı (TAS - True Airspeed) (ft/sec)

- Kullanıcının yapmış olduğu girdiler:
  - Eğitmen konsolundan atmosferik model bilgileri:

    1.Standart havà

    2.Sıcak hava

    3.Tropik hava

    4.Kutup havası

    5.Soğuk hava
  - Seçilen atmosferik hava sıcaklığından sapma sıcaklığı (Deg C).
  - ADC arıza aktivasyonu
- Avionic bilgisayar girdileri:.
  - Barometrik basınç değeri (InHg)

Simulatör sisteminin parçası olan kokpitin çevresinde gerçek uçakta bahsedilen sensörler ve bunları elektriki sinyala dönüştüren devreler mevcut olmadığından istenilen hava şartları yukarıdaki girdi bilgilerinden de görüleceği gibi eğitmenin konsoldan ya da öğrenci pilotun aviyonik bilgisayara kokpit içerisinden bildirmesiyle elde edilmektedir.

Uçuş modeli önceki bölümde de anlatıldığı gibi, uçağın her türlü hareketinin hesaplamalarının yapıldığı, durum bilgisinin oluşturulduğu modeldir. Hazır olarak burada bulunan verilerin modelde kullanılması modelin tüm simulatör sistemi ile uyumlu çalışması açısından önemlidir. Her model bir hesabı ayrı ayrı yapar ise yapılacak binde birlik hata bile tüm sistemin işleyişinde farklılıklara sebep olacaktır. Bu sebeple, normalde modelin hesaplaması gereken deniz seviyesinden yükseklik ve gerçek hava hızı uçuş modelinden elde edilmektedir.

Eğitmen konsolundan girilen atmosferik tip, uçağın bulunduğu yükseklikteki atmosferik sıcaklığı belirlemektedir. Soğuk bir atmosferik ortam ile tropik bir ortamda bulunan aynı yükseklikteki iki uçak farklı iki sıcaklık içerisinde bulunacaktır. Bu sıcaklıkların elde edilmesi ile ilgili olarak bazı atmosferik

tablolar bulunmaktadır. Bu tablolar kullanılarak modelin ihtiyacı olan sıcaklık bilgileri elde edilmektedir.

Hava veri bilgisayar modeli askeri bir proje için hazırlanıyor ise, modelin asgari standartlara uyabilmesi için atmosfer şartlarının aşağıdaki standartlart tablolardan elde edilmesi gerekmektedir.

- é1962 U.S. Standart Atmosfer Tabloları [14]
- MIL-STD 210A Climatic Extremes for Military Equipment [9]

Hazırlanan modelin bu tablolarla uygun olmalarının gerekmesinin sebebi, üretilmiş olan Hava Veri Bilgisayarlarının asgari standartlara uygunluk testlerinin bu tablolara göre yapılıyor ve tüm dünyaca kabul görmüş standart olmasıdır. Bu sebeple atmosfer şartları ile ilgili modelde kullanılacak tabloların bu standartlardaki değerlerden elde ediliyor olması, modelin hazırlanmasında da kolaylık sağlamaktadır.

U.S. Standart Atmosfer Tabloları, Birleşik Devletlere ait Standart Atmosfer Komitesinin (COESA), bir uzantısı olarak 1953 yılında kurulan ekip Standart atmosfer tablolarını oluşturmuştur. İlk tablo 1958 yılında ortaya çıkmış ve sırasıyla 1962, 1966 ve 1976 versiyonları gelişmiştir. Bu dokümanın oluşması için yapılan çalışmalar sırasında NOAA, NASA, U.S. Hava Kuvvetlerinin yanı sıra, Amerkan hükümeti, endüstrisi, araştırma enstitüleri ve üniversitelerini temsil eden otuzu aşkın kuruluşun da yardımları bulunmuştur. Roket ve uydu verileri ile mutlak gaz teoreminin esas oluşturduğu bu çalışmalarda deniz seviyesinden 1000km seviyesine kadar yükseklik içerisindeki tüm seviyelerin atmosferik yoğunluk ve sıcaklık verileri elde edilmiştir.1958, 1962 ve 1976 standart atmosfer tabloları kararlı şartlardaki atmosfer durumları için belirli yüksekliklerdeki sıcaklık, basınç, yoğunluk, yerçekimi ivmesi, yükseklik basıncı, partikül ortalama hızı, ortalama çarpışma frekansı, ortalama moleküler ağırlık, ses hızı, dinamik vizkozite, kinematik vizkozite, termal iletkenlik, jeopotansiyel yükseklik gibi değerleri çıkarmıştır. Yükseklik aralığı, düşük yüksekliklerde 0.05 km iken yüksek yüksekliklerde 5 km'dir. Tüm tablolar ingiliz birim sistemi (feet) ve aynı zaman da metrik (metre) olarak mevcuttur

MIL-STD 210A Climatic Extremes for Military Equipment; dokümanında ise yeryüzünde mevcut tropik, kutupsal, sıcak gün, soğuk gün gibi atmosfer şartları için mevcut standart deniz seviyesi basınç, ses hızı, yoğunluk gibi değerlerin tabloları bulunmaktadır. Seçilmiş olana atmosfer tipi sabit bir sıcaklığa göredir. Seçilen atmosfer şartından farklı bir sıcaklıkta uçuş gerçekleştiriliyor ise, girilen sapma değeri ile onun çevresinde bir değere atama yapılabilmektedir.

Barometrik basınç sabiti istenilen bir noktaya yükseklik hesabı yapabilmek için kullanılmaktadır. Mesela deniz seviyesine göre yükseklik hesaplamak istersek bu sabitin değeri 29,92 InHg olmaktadır. Bu sabit 27 ile 31 InHg arasında değer göstermektedir.

Bu modelin çalışması sonucunda beklenen çıktılar şunlardır:

- Statik hava sıcaklığı (deg R)
- Hava yoğunluğu (slug/ft^3)
- Statik hava basıncı (psf)
- Gerçek hız (ft/sec)
- Mach sayısı (--)
- Dinamik basınç (psf)
- Darbe basıncı (psf)
- Kalibre edilmiş hız (ft/sec)
- Toplam sıcaklık (degR)
- Toplam basınç (psf)
- Hava yoğunluk oranı
- Barometrik düzeltilmiş yükseklik (ft)

4.1.1.Hava veri bilgisayar matematik modeli

Modeli başlangıç durumuna getirmek için seçilen atmosferik model tipine göre deniz seviyesi sıcaklık, basınç, yoğunluk değerleri Çizelge 4.1 ve Çizelge

4.2'ye göre atanmaktadır. Bu fonksiyon her model aktif edildiğinde çağırılmaktadır.

Modelin güncelleme fonksiyonu seçilen atmosferin belirlenen standart verilerine, girilen barometrik düzeltme değeri ve girilen deniz seviyesi değerlerine göre sıcaklık ve basınç oranlarını hesaplamaktadır. Bu matematik modeller Standart Atmosfer Denklemlerinden elde edilmiştir [15]. Daha sonra da sırasıyla aşağıdaki değerlerin hesabını gerçekleştirmektedir.

4.1.1.1. Statik Hava Sıcaklığı (deg R)

Statik Hava Sıcaklığı ($static\_air\_temp$)

$$static\_air\_temp = sea\_level\_temp \times temp\_ratio \qquad (4\text{-}1)$$

formulüyle elde edilmektedir.

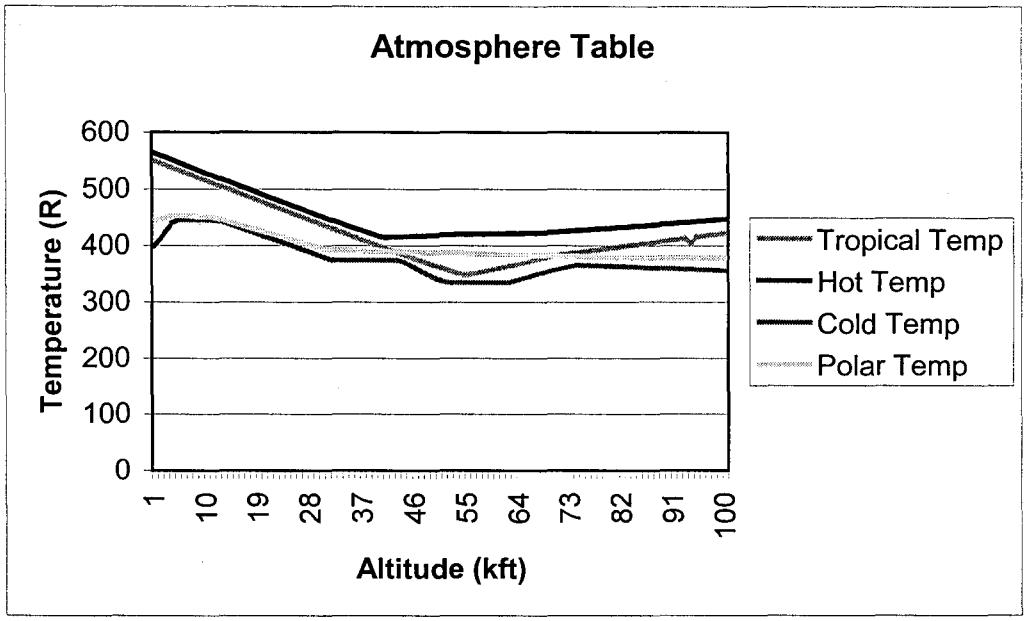
Burada (sea_level_temp) deniz seviyesi sıcaklığı olup seçilen atmosfer tipine göre değişim gösteren sabittir. Bu sabitler, MIL-STD 210A Climatic Extremes for Military Equipment dokümanındaki mevcut tablolardan elde edilmektedir.

Bu model için ihtiyaç duyulan sabitler Çizelge 4.1 ve Çizelge 4.2'de verilmiştir.

Çizelge 4.1 MIL-STD 210A Climatic Extremes for Military Equipment'a göre çeşitli atmosfer şartlarının deniz seviyesi sabitleri

	STANDARD DAY	HOT DAY	TROPICAL DAY	POLAR DAY	COLD DAY
Sea Level Temperature	518,69	562,70	562,70	444,00	399,70
Sea Level Pressure	2116,22	2116,22	2116,22	2140,70	2116,22
Sea Level Sound	1116,44	1162,84	1162,84	1032,94	980,05
Sea Level Air Density	$2,3769 \times 10^{-3}$	$2,19 \times 10^{-3}$	$2,246 \times 10^{-3}$	$2,817 \times 10^{-3}$	$3,09 \times 10^{-3}$

Çizelge 4.2 é1962 U.S. Standart Atmosfer Tablolarına göre Yüksekliğe göre sıcaklık değişim tablosu



Sıcaklık oranı $(temp\_ratio)$ uçağın bulunduğu yüksekliğe göre değişim göstermektedir, buna göre;

$altitude \leq 36089.0 \Rightarrow$

$$temp\_ratio = \theta = \left(1.0 - 6.87528 \times 10^{-6} \times altitude\right) + \left(temp\_dev / sea\_level\_temp\right) \quad (4\text{-}2\text{-a})$$

$36089 \langle altitude \leq 65824.0 \Rightarrow$

Ts = -56.50°C = 216.65°K sabit

$$temp\_ratio = \theta = 0.7519 + \left(temp\_dev / sea\_level\_temp\right) \quad (4\text{-}2\text{-b})$$

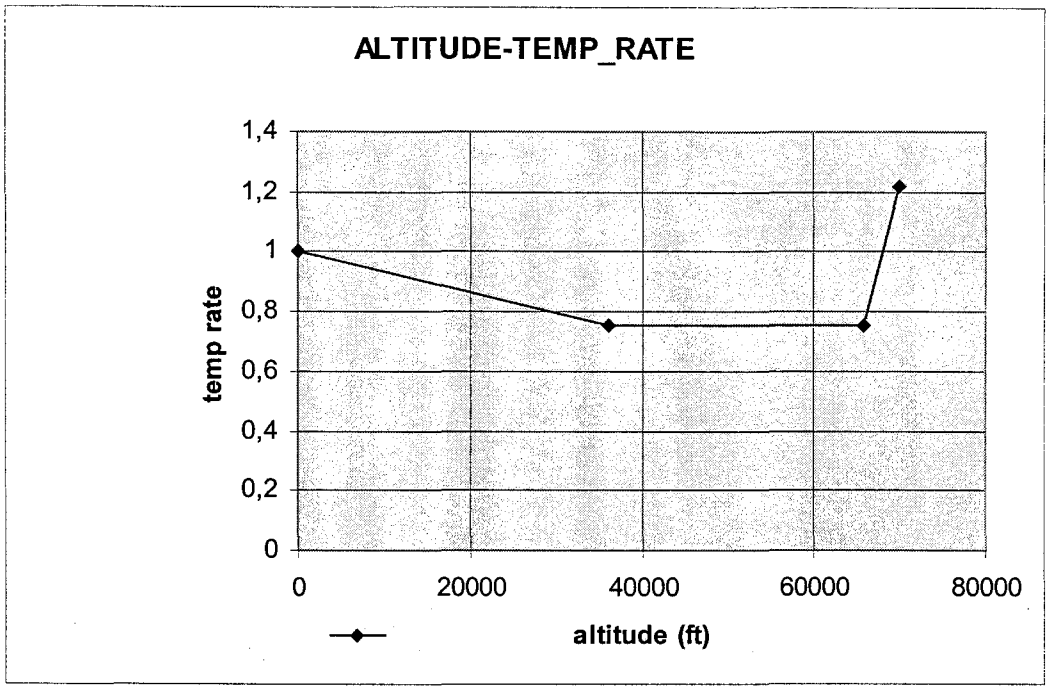
$altitude \rangle 65824.0 \Rightarrow$

$$temp\_ratio = 1.0578 \times 10^{-4} \times \left(altitude - 65616.8\right) + 0.751865 + \frac{temp\_dev}{sea\_level\_temp} \quad (4\text{-}2\text{-c})$$

formulleriyle hesaplanmaktadır.

Yukarıdaki formüller incelendiğinde, yüksekliğe göre sıcaklık oranı Tablo 4.3'deki gibi değişim gösterdiği görülmektedir.

Çizelge 4.3 Yüksekliğe göre sıcaklık oran değişimi



Sıcaklık oran formulünde verilen sıcaklık farkı (temp_dev) kullanıcının seçtiği atmosfer tipindeki standart sıcaklıktan Rankin cinsinden sapma sıcaklığını vermektedir.

4.1.1.2. Statik Hava Basıncı (psf)

Statik hava basıncı $(static\_air\_pressure)$

$$static\_air\_pressure = sea\_level\_pressure \times pressure\_ratio \qquad (4\text{-}3)$$

formülüyle elde edilmektedir.

Burada değerine ihtiyaç duyulan basınç oranı $(pressure\_ratio)$.

$altitude \leq 36089$

$$\Rightarrow pressure\_ratio = (temp\_ratio)^{5,256152} \qquad (4\text{-}4\text{-}a)$$

$36089 \langle altitude \leq 65824.0$

$$\Rightarrow pressure\_ratio = 0{,}223361 \times e^{(36089{,}2 - altitude) \times 4{,}80637 \times 10^{-5}} \quad (4\text{-}4\text{-}b)$$

$altitude \rangle 65824.0 \quad \Rightarrow$

$$pressure\_ratio = 0{,}0540328 \times (0{,}751865 / temp\_ratio)^{34{,}203357} \quad (4\text{-}4\text{-}c)$$

formülüyle hesaplanmaktadır.

Deniz seviyesi basınç $(sea\_level\_pressure)$ verisi tablo 4.1'den elde edilmektedir.

4.1.1.3. Hava Yoğunluk Oranı

İdeal gaz denklemi,

$$P = \rho \cdot g \cdot R \cdot T \quad (4\text{-}5)$$

Burada g yerçekimi sabiti,

R gaz sabitidir.

Bu sabitler denklemin sol tarafına alındıklarında denklem

$$g \cdot R = P / \rho \cdot T \quad (4\text{-}6)$$

şeklinde de ifade edilebilir. Eşitliğin sağ tarafı da sabit olup buradan aşağıdaki formül çıkarılır. Böylelikle hava yoğunluk oranı $(density\_ratio)$

$$density\_ratio = pressure\_ratio / temp\_ratio \quad (4\text{-}7)$$

formulüyle hesaplanmaktadır.

Burada ihtiyaç duyulan $temp\_ratio$ ve $pressure\_ratio$ değerleri sırasıyla eşitlik (4.2) ve eşitlik (4.4) hesaplamalarından elde edilmektedir.

4.1.1.4. Hava Yoğunluğu (slug/ft^3)

Deniz seviyesi hava yoğunluğu $(sea\_level\_air\_density)$ verisi tablo 4.1'den elde edilmektedir. Yoğunluk oranı $(density\_ratio)$ eşitlik (4.7)'de hesaplanmaktadır. Buna göre Hava Yoğunluğu (slug/ft^3) $(air\_density)$

$$air\_density = sea\_level\_air\_densityxdensity\_ratio \quad (4\text{-}8)$$

formulüyle elde edilmektedir.

4.1.1.5. Gerçek Hız (ft/sec)

Uçağın katetmiş olduğu gerçek hız (True Airspeed-TAS) simulatör sistemlerindeki uçuş modelinde hesaplanmakta olup, girdi olarak elde edilen değer, modelin çıktısı olarak da direk kullanılmaktadır.

4.1.1.6. Mach Sayısı (--)

Mach sayısı $(mach\_number)$, uçağın sahip olduğu gerçek hızın bulunduğu yükseklikteki ses hızı değerine oranı olup

$$mach\_number = TAS / sound \quad (4\text{-}9)$$

$$sound = \sqrt{\gamma \times R \times static\_air\_temp} \quad (4\text{-}10)$$

formülüyle hesaplanmaktadır.

Burada, γ, spesifik ısı sabitleri oranı olup 1.4,

R, spesifik gaz sabiti olup 1716 (ft.lb)/(slug. ^{0} R) dir.

Böylelikle ses hızı $(sound)$ sadece statik hava sıcaklığına bağlı olarak

$$sound = 49{,}021 \times \sqrt{static\_air\_temp} \quad (4\text{-}11)$$

formülüyle elde edilmektedir.

4.1.1.7. Dinamik Basınç (psf)

Dinamik basınç $(dynami\_pressure)$

$$dynamic\_pressure = 0{,}5 \times air\_density \times TAS^2 \quad (4\text{-}12)$$

formülüyle hesaplanmaktadır.

Buna göre $(air\_density)$ Eşitlik (4-8)'de hesaplanmaktadır.

4.1.1.8. Darbe Basıncı (psf)

Toplam basınç ile statik basınç arasındaki fark olarak da adlandırılır. Model, kalibre edilmiş hızı hesaplayabilmek için statik ve toplam basınç değerlerinden etki basıncı Q_c, ve görüntülenen etki basıncını Q_{ci} hesaplamaktadır [1].

$mach\_number \leq 1{,}0 \Rightarrow$

$$impact\_pressure = \left(\left(1{,}0 + 0{,}2 \times mach\_number^2\right)^{3{,}5} - 1{,}0\right) \times static\_air\_pressure \quad (4\text{-}13\text{-a})$$

$mach\_number > 1{,}0 \Rightarrow$

$$impact\_pressure = \left(\left(\frac{\left(166{,}9 \times mach\_number^2\right)}{\left(7{,}0 - \left(1{,}0 / mach\_number^2\right)\right)^{2{,}5}} - 1{,}0\right)\right) \times static\_air\_pres. \quad (4\text{-}13\text{-b})$$

4.1.1.9. Kalibre Edilmiş Hız (ft/sec)

Ses hızının geçildiği noktada yapılan kalibre edilmiş ses hızı hesabında yine hesaplanan birimin kendisi bulunmaktadır. Bunun sebebi öncelikle ses hızı geçilmeden önceki hesabın yapılmasının gerekli olmasıdır.

Kalibre edilmiş hız (CAS)

$mach\_number \leq 1{,}0 \Rightarrow$

$$CAS = 2496{,}4357328 \times \sqrt{1{,}0 + \left(\frac{impact\_pressure}{sea\_level\_pressure}\right)^{0{,}285714} - 1{,}0} \quad (4\text{-}14\text{-a})$$

$mach\_number > 1{,}0 \Rightarrow$

$$CAS = \left(51{,}1987 \times \left(\frac{impact\_pressure}{sea\_level\_pressure} + 1{,}0\right) \times \left(7 - \frac{sea\_level\_sound}{CAS}\right)^2\right)^{1{,}25} \quad (4\text{-}14\text{-b})$$

formulleriyle hesaplanmaktadır.

Buna göre $(sea\_level\_pressure)$ ve (sea_level_sound) değerleri MIL-STD 210A standart dokümanından elde edilmiş olan Çizelge 4.1'den, $(impact\_pressure)$ değeri de eşitlik 4-13'den elde edilmektedir.

4.1.1.10. Toplam Sıcaklık (degR)

Toplam sıcaklık $(total\_temp)$

$$total\_temp = static\_temp \times \left(1 + (1+\gamma)/2 \times mach\_number^2\right) \quad (4\text{-}15)$$

formülüyle elde edilmektedir.

Burada, $(static\_air\_temp)$ eşitlik 4-1'den, $(mach\_number)$ ise eşitlik 4-9'dan elde edilmektedir.

4.1.1.11. Toplam Basınç (psf)

Toplam basınç $(total\_pressure)$

$$total\_pressure = static\_air\_pressure + impact\_pressure \quad (4\text{-}16)$$

formülüyle elde edilmektedir.

Burada, $(static\_air\_pressure)$ eşitlik 4-3'den, $(impact\_pressure)$ değeri de eşitlik 4-13'den elde edilmektedir.

4.1.1.12. Barometrik Düzeltilmiş Yükseklik (ft)

Gerçek yüksekliğin elde edilebilmesi için barometrik düzeltme (baro_alt_correction) değerinin kullanıcı tarafından doğru olarak girilmesi gerekmektedir. Bu değer girilmez ise standart olarak deniz seviyesi değeri olan 29,92 değeri ile hesaplamalar yapılmaktadır. Eğer uçağın konumu deniz seviyesinden farklı ise hesaplanan yükseklik değeri doğru olmayacaktır.

Barometrik düzeltilmiş yükseklik $(baro\_corrected\_altitude)$

$$baro\_corrected\_altitude = altitude + delta\_altitude \quad (4\text{-}17)$$

formülüyle elde edilmektedir.

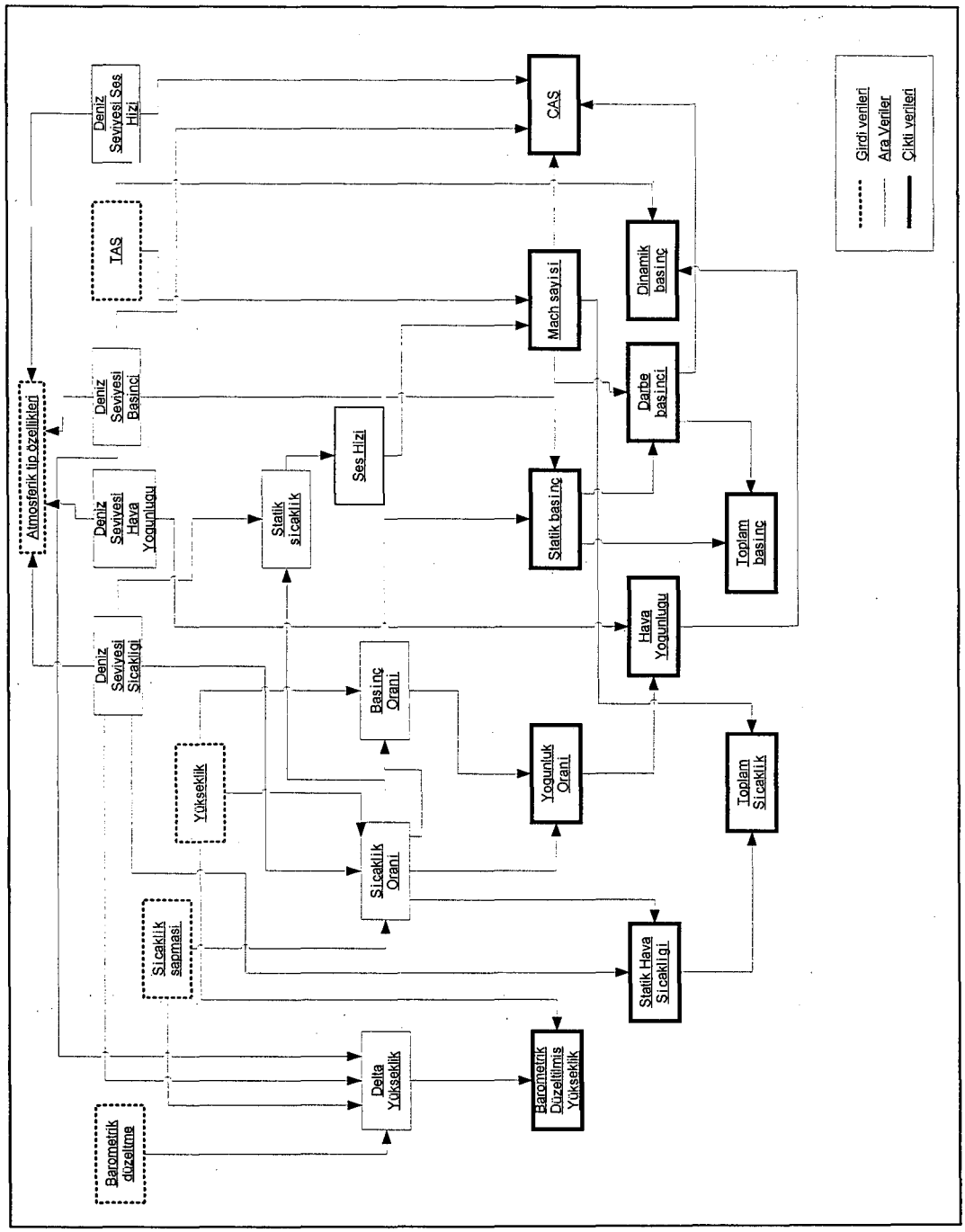
Burada $(delta\_altitude)$

$$delta\_altitude = \frac{sea\_level\_temp + temp\_dev}{0{,}00356616} \cdot X$$

$$X = \left(\left(\frac{baro\_alt\_correction \times hg\_2\_psf}{sea\_level\_pressure}^{\frac{-1{,}0}{-5{,}25612}} - 1{,}0 \right) \right) \quad (4\text{-}18)$$

formulüyle hesaplanmaktadır.

Barometrik düzeltmenin etkilediği değerlerin hangileri olduğu ve atmosferik şartlara, standart ve standart olmayan gün şartlarına bağlı olarak hesaplanan atmosferik değerler veri ilişkilerinin verilmiş olduğu Tablo 4.4'de gösterilmiştir.



Çizelge 4.4 ADC Model Veri İlişki Diagramı

Aviyonik sistemlerde, özellikle hava veri bilgisayarı ve ataletsel seyrüsefer sistemleri gibi verileri sürekli kullanılan ünitelerden elde edilen verilerin merkezi bir bilgisayara gönderilmesi için mux-bus haberleşme protokolü kullanılmaktadır. Aviyonik sistemlerin merkezi bir bilgisayar tarafından kontrol edilmesinin sebebi, bu bilgisayarın seyrüsefer ve silah salımları ile ilgili hesaplamaları yapıyor olması ve hesapladığı seyrüsefer ve silah salım yardımcı işaretlerini çeşitli göstergelerde göstermesidir. Bu işlemleri yapabilmesi için tüm aviyonik sistemlerden veriler önemlerine göre önceden planlanmış sıklıkla elde edilmekte ve gerekli sistemlere istenilen formatlara dönüştürülerek yollanmaktadır.

4.1.2.MIL-STD-1553B mux-bus haberleşme ağı

Uçak aviyonik sistemlerinin arayüzü olarak yaygın bir şekilde kullanılan Mux-Bus haberleşme ağı ile üniteler arasındaki haberleşmelerin önemli bir bölümü gerçekleştirilmektedir. Bu standart ile birlikte aviyonik sistemler arasında merkezi bir kontrol mekanizması oluşturulmuş, standart arabirimler meydana getirilmiş, kablo ve konnektör sayısı azaltılmış, fiziksel tıkanıklık ortadan kaldırılmış, ağırlık azaltılmış, Electro Magnetic Compalibility (EMC) uygunluğu arttırılmıştır. Sistemin tasarım, onarım ve geliştirme imkanı daha basit ve ucuz hale getirilmiştir.

1968 yılında, SAE A2K komitesi, endüstri kuruluşları ile toplanarak *multiplexed* sistemi tanımlayan bir askeri standart oluşturmuştur. 1973 yılında MIL-STD-1553 ve 1978 yılında da MIL-STD-1553B yayınlanmıştır. Bu standart veri yolunun tüm karakteristiklerini, iletim tekniklerini, iletim protokolünü, terminal arabirim karakteristiklerini ve kullanılan ekipmanların değiştirilebilirlik şartlarını - hangi firmanın ürettiğine bakılmaksızın – tanımlamaktadır.

Mux-Bus protokolü bir seri haberleşme sistemidir. Tüm bilgi bitleri mesaj blokları halinde ardışıl olarak aktarılır. Protokolde kullanılan bilgi aktarım hattı bir tanedir ve en fazla 32 adet kullanıcı ünite bu hat üzerine bağlanmaktadır. Bilgi hattına bağlanmış üniteler zaman paylaşımı ile bu hattı sırasıyla kullanmaktadırlar. Sistem kontrolcüsü tarafından gönderilen komutlara göre ilgili üniteler cevap verirler ve yapılması gereken bilgi transferi gerçekleştirilir. Bu sisteme *speak when spoken to* denmektedir [3].

Aviyonik Mux-Bus protokolü 3 tip elemandan oluşur [16]. Bunlar :

- Bus Controller (BC)
- Bus Monitor (BM)
- Remote Terminal (RT)

Bus Controller (BC) tüm veri yolu operasyonun yönetir ve kontrol eder. Hatta bağlanmış olan ünitelere komutlar göndererek onları aktif hale getirir [17]. Aviyonik sistemlerde bu görevi Görev Bilgisayarı gerçekleştirmektedir.

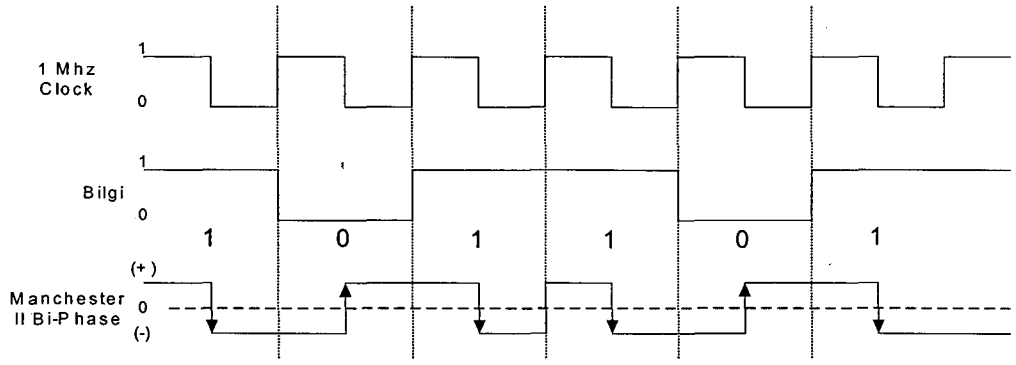
Bus Monitor, Mux-Bus hattında dolaşan tüm bilgileri göstermek veya kaydetmek için kullanılan bilgisayar tabanlı ünitedir. BM, isteğe bağlı olarak sisteme eklenen bir ünitedir [17].

Mux-Bus hattında Bus Controller dışında bilgi iletimi yapan tüm ünitelere Remote Terminal (RT) denir. (örneğin RADAR, EGI, ADC vs.) Kurulacak olan bir Mux-Bus yapısında tüm RT'lere bir adres (numaralandırma) verilmesi gerekmektedir. Ancak BC ve BM birer RT olmadıkları için RT adreslemeleri bulunmamaktadır. Bu adresler RT lere donanım üzerinden, BC ünitesine de yazılım vasıtasıyla bildirilir. Sonuçta her RT bir adres alır ve böylece BC hangi RT ile haberleştiğini ve hangi verinin hangi RT için olduğu bilinmektedir.

Mux-Bus haberleşme yapısı şu 3 kelimeyle özetlenebilir:

- Seri
- Zaman paylaşımlı (time division)
- Komut / Cevap bilgi veri yolu

MIL-STD 1553B protokolü, seri dijital *pulse code modulation (PCM)* modülasyonunu kullanarak *Manchester II Bi-Phase* seviyesi bilgi kodlama yöntemiyle bilgi aktarımı yapar. Bilgi aktarım hızı saniyede 1Mbyte tır. Manchester II Bi-Phase kodlama örneği şekil 4.1'de görülmektedir.



Şekil 4.1 Manchester II Bi-Phase dijital kodlama yöntemi

Sistemdeki clock sinyali BC tarafından sağlanır. Tüm RT ler bu zaman senkronizasyonuna göre çalışırlar. Aviyonik sistemlerde bir Mux-Bus döngü süresi (cycle) 50ms dir. 50ms lik zaman dilimleri içinde tüm bilgi aktarımları tamamlanır ve bir sonraki döngü işleme girer [16].

MIL-STD 1553B standardına göre bir Mux-Bus seri iletişim hattında 3 tip word yapısı bulunmaktadır [16, 17].

- Command Word
- Status Word
- Data Word

4.1.2.1. Command Word Formatı

BC den RT lere veri yolu sistemini etkin hale getirmek için gönderilen word dür. Veri yolundaki her bilgi akışı değişiminde BC den ilgili RT lere bu word gönderilir[16, 17].

4.1.2.2. Data Word Formatı

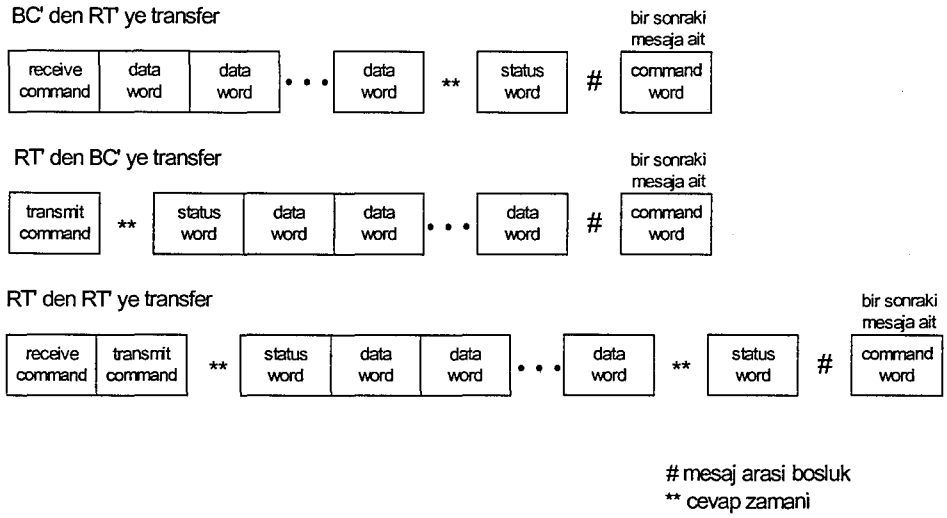
BC ile RT lerden, command word veya status wordden sonra gönderilen bilgileri taşıyan word yapısıdır. (Şekil 4.3) [16, 17].

4.1.2.3. Status Word Formatı

Bu word BC tarafından herhangi bir RT ye gönderilen command word'e ilgili RT nin cevabını bildiren word'dür [16].

Mux-bus hattı üzerinde üç çeşit veri transferi olmaktadır (Şekil 4.2).

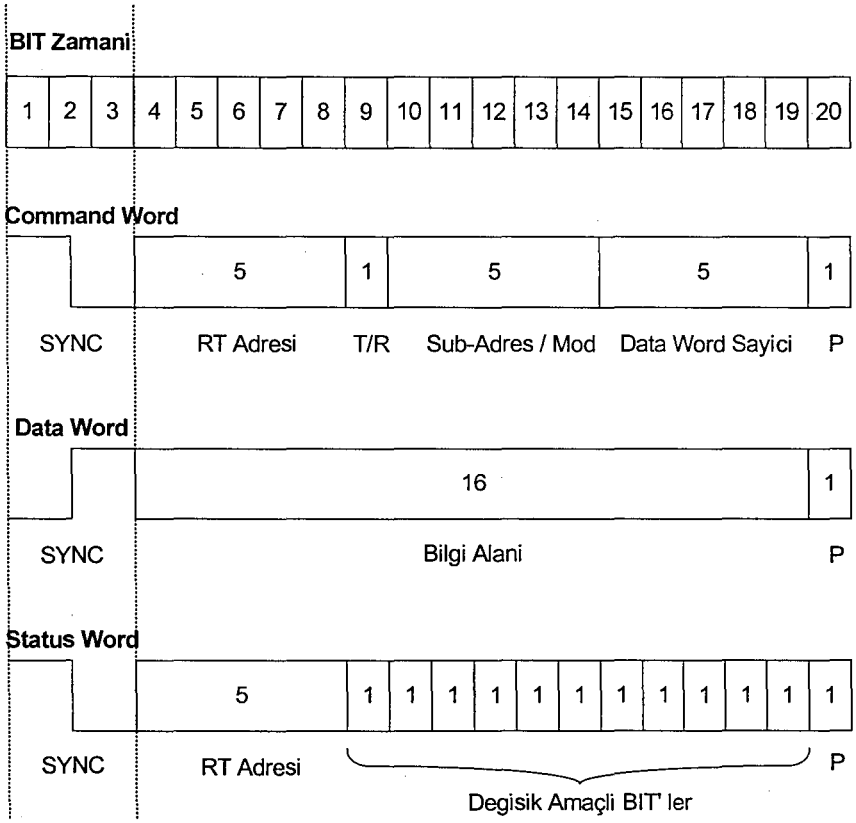
- BC'den RT'ye veri transferi
- RT'den BC'ye veri transferi
- RT'den RT'ye veri transferi



Şekil 4.2 Mux-Bus Word yapıları

4.1.2.4. Mux-Bus Mesaj Formatları

Bir aviyonik mux-bus yapısında command word, status word ve data wordlerden oluşan birleşime mux-bus mesaj bloğu denir. Mesaj blokları BC tarafından gönderilen command word ve arkasından gelen en fazla 32 adet data word şeklinde olabileceği gibi, RT ler tarafından gönderilen status word ve arkasından gelen en fazla 32 adet data word bileşimi şeklinde de olabilir. 2 tip mesaj bloğu haberleşmesi vardır. Bunlar;



Şekil 4.3 Mux-bus mesaj haberleşmesi

- BC ile RT arası haberleşme
- RT de RT ye haberleşme olarak tanımlanır.

4.1.2.5. BC ile RT Arası Haberleşme

BC ile RT ler arasındaki bilgi iletişimi 2 şekilde yapılır. Command Word de bulunan T/R bitine göre BC ya RT lere bilgi gönderir veya RT lerden bilgi alır.

4.1.2.6. RT den RT ye Haberleşme

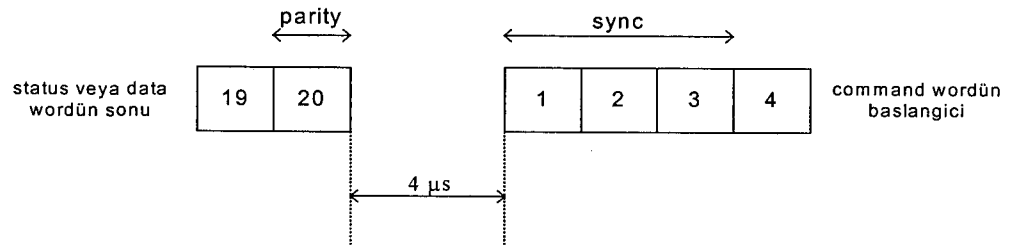
Mux-Bus sisteminde BC den RT lere haberleşme yapıldığı gibi, RT lerden RT lerede haberleşme imkanı vardır. Bu haberleşme de BC tarafından kontrol edilir.

RT#1 ile RT#2 arasında. RT#1'in RT#2'ye bilgi göndermesi gerektiğinde. ilk olarak BC, RT#2 ye bir *receive command word* (T/R = 0), RT#1 e de *transmit command word* (T/R = 1) gönderir. Bilgiyi gönderecek olan RT#1, BC ya bir status word, arkasındanda RT#2 ye gerekli data word'leri yollar. Son olarak,

RT#2 bilginin alınıp alınmadığına dair BC ye bir status word gönderir. Böylece RT den RT ye bilgi transferi, BC yönetiminde gerçekleştirilmiş olur.

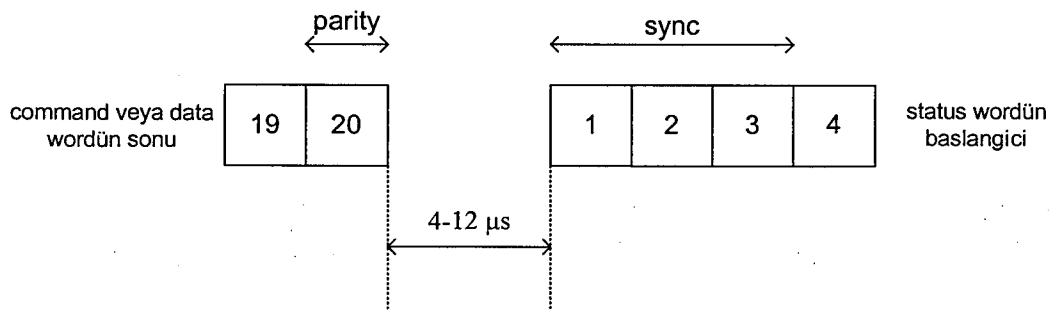
4.1.2.7. Blok Zaman Aralıkları

Mux-bus mesaj bloğu haberleşmesinde, BC'nin RT'den aldığı son mesaj bloğundan sonra yeni bir mesaj göndermesi için geçmesi gereken süre 4 μs dir Şekil (4-4) [16].



Şekil 4.4 BC mesaj üretme süresi

RT tarafından BC den gelen bir mesaj bloğu alındığında buna cevap mesajı göndermesi için geçmesi gereken süre 4-12μs dir Şekil (4-5), [16].



Şekil 4.5 RT mesaj üretme süresi

Bir aviyonik mux-bus yapısı basit ve fonksiyonel olmak üzere iki şekilde tasarlanabilir. Basit yapı; bir BC, Backup Bus Controller (BBC) özelliği olan bir RT ve mux-bus hattına bağlı en fazla 32 adet olabilecek RT lerden meydana gelir. Sistemin ana kontrolcüsü olan BC ünitesi en fazla iki mux-bus hattını kontrol edebilir ve her iki hatta toplam olarak 64 RT bulunabilir. BC de herhangi bir arıza

oluştuğunda sistemin kontrolünü mux-bus kontrol arayüzü adı verilen bir sinyal vasıtasıyla BBC alır.

Ünite (BC veya RT) Mux-Bus arasındaki tüm yazılım ve donanım gereksinimlerinin yapıldığı modüle Mux-Bus Interface (MBI) adı verilir. Bu modül mikro-kontrolcü tarafından kontrol edilen standart bir karttır. MBI vasıtasıyla ünitenin tüm haberleşme işlemi gerçekleştirilmektedir.

4.1.2.8. RT Adres Tanımlama Sinyalleri

Aviyonik sistemlerde Mux-Bus arayüz hattına sahip olan tüm RT ler, ünitenin RT adreslerini RT kesikli sinyalleri sayesinde bilirler. RT adres arayüz hattı 6 kesikli GND/OPEN sinyalden oluşur. İlk sinyal referans GND sinyalidir. (LSB) A0, A1, A2, A3 ve A4 (MSB) sinyalleri ise GND=Lojik 1 ve OPEN=Lojik 0 olan ikili tabandaki dijital verilerdir. Bir projede kullanılacak olan RT lerin adresleri tasarımcı sistem mühendisleri tarafından belirlenir. Örneğin RT adresi 5 olan bir mux-bus RT ünitesi için $(5)_{10} = (00101)_2$ olarak tanımlanır (Çizelge 4.5) . Buna göre RT adres kesikli sinyallerinin konumları şöyle oluşur.

Çizelge 4.5 RT adres 5 için RT adres kesikli sinyallerinin durumları

A0	A1	A2	A3	A4
1	0	1	0	0
GND	OPEN	GND	OPEN	OPEN

Simulatör sistemlerinde gerçek ünite kullanılmadığı durumlarda yukarıda bahsedilen adresleme ünite olmadığı için yazılan model içerisinde tanımlaması yapılmaktadır. Modelin yüklü olduğu bilgisayara da MBI kartı takılarak bu veri yolu ile haberleşme işlemi gerçekleştirilebilmektedir.

4.1.3.Hava veri bilgisayarı yazılım modeli

C++ dilinde yazılmış olan modelin kaynak kodları Ek-1 de verilmiştir. Bu modelin sahip olduğu sınıflar ve bunların yapıları ana hatlarıyla aşağıda anlatılmaktadır. Buna göre;

```
typedef struct init_adc_input
 {
  int    Atmos_Type;
  double Temp_Dev_DegC;
  double Altitude;
 }Init_Adc_input;
```

Buna göre Hava veri bilgisayar modelinin initialize edilebilmesi için yani modelin çalışma öncesi hazırlık işlerinde kullanılan girdiler `Init_Adc_input` yapısı altında tanımlanmıştır. Yukarıda tanımlaması gerçekleştirilmiş hazırlık sırasındaki girdi tiplerinin kısa açıklamaları aşağıda verilmiştir:

`Atmos_Type`	ADC modelinin çalışması istenen atmosfer tipini içermektedir. Buna göre 1- Standart Day 2-Hot Day 3-Tropical Day 4-Polar Day 5-Cold Day atmosfer tiplerinden biri değerlendirmeye alınmaktadır
`Temp_Dev_DegC`	ADC modelinin çalışması için seçilmiş olan atmosfer tipinin sahip olduğu sıcaklıktan olan sapma değeri (derece C) için kullanılmaktadır.
`Altitude`	Sistemin hazırlanması sırasında deniz seviyesinden olan yükseklik değerinin elde edilmesi için kullanılmaktadır.

ADC modelinin çalışması için ihtiyaç duyulan girdiler için Adc_Input tip tanımlaması ve çalışma sonucu üretilen çıktılar için de Adc_Output tip tanımlaması yapılmıştır. Matematik modelinde de bahsedildiği gibi girdilerden çıktılar elde edilirken atmosferik bilgiler ile ilgili veriler için Atmos_data tip tanımlaması yapılmıştır.

```
typedef struct adc_input
 {
  double Altitude;
  double TAS;
  double BaroAltitudeCorrection;
 }Adc_Input;
```

Girdi tipini oluşturan Adc_Input elemanlarının detaylı açıklamaları modelin teorik açıklama kısımı incelenebilir. Bu elemanların kısa açıklamaları ise şöyledir:

Altitude	Modelin çalışması sırasında deniz seviyesinden olan yükseklik değerinin elde edilmesi için kullanılmaktadır
TAS	Uçuş modelinden elde edilen gerçek hava hızı değeridir.
BaroAltitudeCorrection	Barometrik basınç sabiti istenilen bir noktaya yükseklik hesabı yapabilmek için kullanılmaktadır.

```
typedef struct atmos_data
 {
  int     Atmos_Type;
  double  DeltaTempDevRatio;
  double  SeaLevelTemp;
  double  SeaLevelPressure;
  double  SeaLevelAirDensity;
  double  SeaLevelSound;
  double  SeaLevelTemp_no_dev;
 } Atmos_data;
```

Atmos_data, atmosfer ile ilgili bilgilerin atamalarının gerçekleştirildiği yapıdır. Seçilen atmosferik tipe göre yapı içerisindeki elemanların değer atamaları model içerisinde yapılmaktadır.

```
typedef struct adc_output
 {
  double  Temp_static;
  double  Pressure_static;
  double  AirDensity
  double  TAS
  double  Mach;
  double  DynamicPressure;
  double  Veas;
  double  ImpactPressure;
  double  Vcas;
  double  TotalTemp;
  double  TotalPressure;
```

```
            Adc_Output Output;
            Adc_Input Input;
            Adc_Input TestInput;
            Init_Adc_input InitBlock;
            TestMode_Type testMode;
            int testTimer;
    public:
            C_ADC_Model();
            ~C_ADC_Model();
            void Init();
            void Run();
            void CheckActivity();
            void ReadInputs();
    void ResetTestTimer(){ testTimer = 30; };
    void DecrementTestTimer(){ if( testTimer>0 ) testTimer--; }
    void UpdateAdcMainOutMember (Adc_Output *Output);
    void SetAdcInput (Adc_Input *Input);
    void SetAdcSelfTest(Adc_Input *Input,Adc_Output *Output);
    int ADC_UPDATE(Adc_Input IN,Adc_Output *OUT);
    };
```

Model sınıfında kullanılan *private* değişkenlerin açıklamaları aşağıdadır:

`Output`	Yazılım modeli içerisinde Output elemanlarının değişimlerini elde etmek için kullanılmaktadır.
`Input`	Yazılım modeli içerisinde Input

	elemanlarının değişimlerini elde etmek için kullanılmaktadır.
`TestInput`	Test butonunun durumunu tutmak için kullanılmaktadır.
`InitBlock`	Başlangıç değerlerinin tutmak için kullanılmaktadır.
`TestMode`	Gerçekleştirilecek test modunu tutmak için kullanılmaktadır.
`TestTimer`	Test için gerekli sürenün tutulduğu değerdir.

Hava veri bilgisayarının modellenmesi için kullanılan fonksiyonlar aşağıdaki şekilde özetlenmiştir:

`Init()`	Modelin çalışma öncesi hazırlık işleri için kullanılacak model fonksiyonudur.
`Run()`	CADC model çevrimlerinin tek bir defa çalıştırılmaları için yazılmış model fonksiyonudur.
`CheckActivity()`	Modelin çalışmasına engel bir arızanın kullanıcı tarafından oluşturulup oluşturulmadığını belirlemek için yazılmış bir model fonksiyonudur.
`ReadInputs()`	Kullanıcının yapmış olduğu girdilerin değerlendirilmesi için yazılmış model fonksiyondur.
`ResetTestTimer()`	Testin gerçekleşme süresinin belirlendiği fonksiyondur.
`DecrementTestTimer()`	Test moduna girildikten sonra test süresinin

	azaltıldığı fonksiyondur.
`UpdateAdcMainOutMember()`	Model çıktılarının güncellenmesini sağlayan fonksiyondur.
`SetAdcInput ()`	ADC girdilerinin güncellendiği fonksiyondur.
`void SetAdcSelfTest()`	ADC modelinin test işleminin gerçekleştirildiği fonksiyondur.
`ADC_Update()`	ADC modelin normal çalışma modunda çalışmasını sağlayan, matematik modeldeki hesaplamaları gerçekleştiren fonksiyondur.

4.2.Yükseklikölçer Modeli

Yükseklikölçer ünitesi önceden de bahsedilgiği gibi uçak havada iken yere sinyal yayınlayarak yükseklik bilgisini elde etmektedir. Simulatör sistemlerinde bunun gerçeleştirilmesi mümkün değildir. Bu durumda bu ünitenin simulasyonu kokpit içerisinden gerçekleştirilmiş anahtar değişikliklerine göre ve uçağın içinde bulunduğu durum bilgisi uçuş modelinde elde edilerek gerçekleştirilecektir. Buna göre, modelin çalışması için ihtiyacı olan girdi bilgileri ve bu bilgilerin edinildiği simulatör sistemleri şunlardır;

- Uçuş modelinden elde edilen girdiler:
  - İrtifa basıncı
  - Yer seviyesinden yükseklik
- Aviyonik bilgisayar girdileri:.
  - Yükseklikölçer kontrol sinyalleri
- Eğitmen Konsolundan:
  - Yükseklikölçer arıza aktivasyonu

Bu modelin çalışması sonucunda beklenen çıktılar şunlardır:

- Yükseklikölçer yükseklik bilgisi
- Uçak yunuslama bilgisi
- Uçak yalpa bilgisi

Bu model tamamıyla kullanıcın yapmış olduğu kontrollere göre çalışmaktadır. Pilotun yapmış olduğu kontrol değişikliklerine göre uçuş modelinden bilgiler kullanılmaktadırlar.

4.2.1.Yükseklikölçer matematik modeli

Kokpit içerisinde yükseklikölçerin çalışmasını kontrol eden bir anahtar bulunmaktadır. VME girdi/çıktı kartlar üzerinden bu anahtarın konumu okunarak çalışma moduna karar verilmektedir. Pilotun uçuş sırasında gerçekleştirmiş olduğu manevralar da değerlendirilerek yükseklikölçerin modeli meydana getirilmektedir.

Buna göre model içerisinde her bir mod modelin bir fonksiyonu olarak değerlendirilebilir.

Test Modu, kokpit içerisinden bu modun aktif edildiği bilgisi elde edildiğinde, eğitmen konsolundan herhangi bir arıza aktif edilmemişse, model Mux-Bus üzerinden kullanılan ünitenin spesifikasyonlarına göre belirlenmiş süre boyunca sabit bir yükseklik bilgisi gönderilir.

Bekleme Modu herhangi bir yükseklik bilgisi gösterimi gerçekleştirilmez. Ünite çalışmaya hazır hale getirilir. Bu mod anahtar ile aktif edildikten sora normal moda geçildiğinde, ünitenin hazır hale gelmesi için beklenilmez. Ünitenin hazır hale gelmesi süreci (warm up) de bir mod olarak modelde kabul edilmiştir.

Normal modda iken, eğitmen konsoldan herhangi bir arızanın aktif olup olmadığına bakılır. Aktif olan arıza var ise, arızanın çeşidine göre model göstergelerde çıktısını oluşturur. Örneğin, mux bus iletişimi kesilmiş ve aviyonik bilgisayara bilgi iletilemiyor ise herhangi bir yükseklik bilgisi gösterilmemekte ya da radar altimetre ünitesinde takılma arızası oluşmuş ise hep takıldığı andaki yükseklik bilgisini gösterme gibi.

Aktif olan arıza yok ise uçağın davranışı kontrol edilir, yani uçuş modelinden alınan yunuslama ve yalpa bilgileri antenin görüş limiti dahilinde ise model, uçuş modelden elde etmiş olduğu yükseklik bilgisini aviyonik bilgisayara göstergelere sürmesi için gönderir.

Eğer uçağın bulunduğu yükseklik, yüksekliköIçer ünitesinin gösterebileceği limitten yüksek ise o zaman yine yükseklik gösterimi gerçekleştirilmemektedir.

4.2.2.YüksekliköIçer yazılım modeli

C++ dilinde yazılmış olan modelin kaynak kodları Ek-2 de verilmiştir. Bu modelin sahip olduğu sınıflar ve bunların yapıları ana hatlarıyla aşağıda anlatılmaktadır. Buna göre;

```
#define C_WARMUP_TIME 50
#define C_IBIT_TIMER 50
```

Daha önceden de bahsedildiği gibi, radar altimetrenin ünitenin normal çalışma şekline göre bir ısınma süresi (Warm up time) bulunmakta ayrıca istenilen zamanda ünite test (IBIT) edilebilmektedir. Bu iki olay için geçmesi gereken süre zarfında ünite yükseklik bilgisi üretmemektedir. Bu süreyi tanımlamak için model içerisinde zamanlayıcılar kullanılıyor olup, bu zamanlayıcıların periyodları yukarıdaki tanımlamalar ile verilmektedir. Genel bir yüksekliköIçer modeli tasarımı olduğu ve kullanılan üniteye göre bu sürelerin farklılık göstermesi sebebi ve ayrıca yazılan modelin daha anlaşılır olması için gerekli süreler hazırlanmış olan modelde yukarıdaki şekilde ısınma (C_WARMUP_TIME) ve test süreleri (C_IBIT_TIMER) olarak tanımlanmıştır.

Yazılım modeli içerisinde yüksekliköIçer çalışma modlarının tanımlanması amacıyla aşağıda belirtilmiş olan CARAMode_Type tip tanımlaması yapılmıştır. YüksekliköIçer çalışma modu değişkeni bu tip ile tanımlanmış, atamalar ve kullanım bu çerçevede içerisinde yapılmıştır. YüksekliköIçerin her bir çalışma moduna özgün davranış biçimi yazılım modeli içerisinde de simüle edilmiştir.

```
typedef enum {
        caramode_OFF = 0,
        caramode_STANDBY = 1,
        caramode_NORMAL = 2,
        caramode_WARMUP = 3
        caramode_IBIT = 4
} CARAMode_Type;
```

Yüksekliköļerin kokpit içerisinde bulunan açma/kapama anahtarının yazılım modelinde kullanılması için aşağıda belirtilen CARASwitch_Type tip tanımlaması yapılmıştır. Bu tanımlamadaki değerler yüksekliköļer anahtarının fiziksel konumlarına karşılık gelmektedir. Bu konum değişikliklerine bağlı mod değişimleri ise bir önce bahsetmiş olduğumuz mod tip değişkeni ile takip edilmektedir.

```
typedef enum {
        caraswitch_OFF = 0,
        caraswitch_STANDBY = 1,
        caraswitch_NORMAL = 2
} CARASwitch_Type;
```

Kokpit içerisinde bulunan ve yüksekliköļeri ilgilendiren iki temel amaçlı anahtar için aşağıdaki OnOff_Type tip tanımlaması yapılmıştır. Sözkonusu anahtarlar '*Master Avionics*' ve '*CARA Power*' anahtarlarıdır.

```
typedef enum {
        S_OFF = 0,
        S_ON = 1
} OnOff_Type;
```

Yazılım modelinde simüle edilmesi planlanan arıza tipleri de aşağıdaki CARAMalfunction_Type tip tanımlaması ile belirtilmiştir. Herhangi bir arıza bulunmaması durumu {*malf_NONE*}, genel iletişim arızası için {*malf_CARA_COMM*}, yükseklik değerinin güncellenememesi arızası için {*malf_CARA_STUCK*}, yükseklik değerinin üretilememesi arızası için {*malf_CARA_ALT_FAIL*} arıza durumları tanımlanmıştır.

```
typedef enum {
        malf_NONE = 0,
        malf_CARA_COMM = 1,
        malf_CARA_STUCK = 2,
        malf_CARA_ALT_FAIL = 3
} CARAMalfunction_Type;
```

Yükseklikölçer modelinin çalışması için ihtiyaç duyduğu girdiler için caraInput tip tanımlaması ve çalışma sonucu üretilen çıktılar için de caraOutput tip tanımlaması yapılmıştır.

```
typedef struct {
        CARAMode_Type caraModeRequest;
        bool ibitRequest;
        unsigned int altLowReferenceRequest;
        CARASwitch_Type caraSwitch;
        OnOff_Type masterSwitch;
        CARAMalfunction_Type caraMalfunction;
        OnOff_Type caraPower;
} caraInput;
```

Girdi tipini oluşturan elemanların detaylı açıklamaları için yükseklikölçer modelinin teorik çıklamasına bakılabilir. Kısa bir açıklama aşağıda verilmiştir:

`caraModeRequest`	Yükseklikölçer modelinin geçmesi istenilen mode değerini içerir. Mod değişiklik talepleri için kullanılır.
`ibitRequest`	Modelin IBIT moduna geçmesi istenildiğinde kullanılır.
`altLowReferenceRequest`	Alçak irtifa referans değeri değiştirilmek istendiğinde kullanılır. Buraya verilen değer, yeni alçak irtifa referans değeri olması istenen yüksekliktir.
`caraSwitch`	Kokpit içerisinde yer alan fiziksel yükseklikölçer açma/kapama anahtarının modellenmesi için kullanılır.
`masterSwitch`	Kokpit içerisinde yer alan Master Avionics anahtarının modellenmesi için kullanılır.
`caraMalfunction`	Yükseklikölçer modelinin simüle edebileceği arızaları girmek için kullanılır.
`caraPower`	Kokpit içerisinde yer alan yükseklikölçer takat anahtarının modellenmesi için kullanılır.

```
typedef struct {
        bool altitudeInvalidSignal;
        bool altitudeLowSignal;
        bool altimeterFailSignal;
        bool ibitInProgressSignal;
```

```
        int radarAltitude;
        int altitudeRate;
        unsigned int altitudeLowReferenceValue;
} caraOutput;
```

Çıktı tipini oluşturan elemanların kısa açıklamaları aşağıda verilmiştir.

`altitudeInvalidSignal`	*radarAltitude* çıktı değişkeninde tutulan yükseklik değerinin geçerli bir değer olup olmadığını bildirmek için kullanılır. Uçağın açısal konumuna bağlı olarak radar ekosu alınamayan durumlar için geçerlidir
`altitudeLowSignal`	Anlık yükseklik değerinin alçak irtifa referans değerinin altına indiğini bildirmek için kullanılır.
`altimeterFailSignal`	Yükseklikölçerin geçerli yükseklik bilgisi üretemediğini bildirmek için kullanılır. Genel bir arızaya işaret eder.
`ibitInProgressSignal`	Yükseklikölçerin test modunda olduğunu, testin devam etmekte olduğunu gösterir.
`radarAltitude`	AltitudeInvalidSignal ve AltimeterFailSignal değişkenlerine bağlı olarak, hesaplanan yükseklikölçer değerini içerir.
`altitudeRate`	Yüksekliğin değişim hızını içerir.
`altitudeLowReferenceValue`	Mevcut alçak irtifa referans değerini içerir.

Modelin çalışması için ihtiyaç duyulmakla beraber girdi ya da çıktı olmayan diğer değişkenler, model sınıfının *private* değişkenler bölümünde tanımlanmıştır. Model sınıfı C_CARA_Model aşağıda verilmiştir.

```
class C_CARA_Model {
        private:
                CARASwitch_Type prevSwitchState;
                CARAMode_Type caraMode;
                int warmupTimer;
                int ibitTimer;
                bool fCARAInWarmup;
                bool fIBITRequested;
                int defaultRadarAltitude;// 300 ft
                int defaultAltitudeRate;// 0 ft/s
                float flight_altitude;
                float flight_alt_rate;
                float flight_pitch;
                float flight_roll;
                void updateModelInput();
                void updateModelOutput();
                void updateCARAMode();
                void caraCycle();
                void warmupCycle();
                void ibitCycle();
                void caraCalculate();
                void startWarmup();
                void endWarmup();
        public:
```

```
                caraInput modelInput;
                caraOutput modelOutput;
                C_CARA_Model();
                ~C_CARA_Model();
                void Run();
                void Init();
                void startIBIT();
                void endIBIT();
};
```

Model sınıfında kullanılan *private* değişkenlerin açıklamaları aşağıdadır:

`prevSwitchState`	Yazılım modeli içerisinde, yükseklikölçer mod anahtarının konumundaki değişimleri algılamak için kullanılır.
`caraMode`	Yükseklikölçerin modelinin çalışma modunu tutmak için kullanılır.
`warmupTimer`	Modelin warmup zamanını modellemek için kullanılan zamanlayıcı değişkenidir.
`ibitTimer`	Modelin test modunda kaldığı zamanı modellemek için kullanılan zamanlayıcı değişkenidir.
`fCARAInWarmup`	Modelin warmup modunda olup olmadığını takip etmek için kullanılır.
`fIBITRequested`	Test moduna geçilip geçilmeyeceğine karar vermek için kullanılır.
`defaultRadarAltitude`	Test sırasında kullanılan default radar yükseklik değeridir.

`defaultAltitudeRate`	Test sırasında kullanılan default yükseklik değişim hızıdır.

Aşağıdaki değerler uçuşun modellenmesi için kullanılır, direkt olarak yükseklikölçer modeliyle ilgili değildir:

`flight_altitude`	Uçuş irtifa değeridir.
`flight_alt_rate`	Uçuş irtifası değişim hızıdır.
`flight_pitch`	Uçağın, kanatları boyunca uzanan eksen etrafında, burun yukarı pozitif olmak üzere, yatay konumdan sapma açısıdır.
`flight_roll`	Uçağın, gövdesi boyunca uzanan eksen etrafında, sağa yatış pozitif olmak üzere, yatay konumdan sapma açısıdır.

Yükseklikölçerin modellenmesi için kullanılan *private* ve *public* fonksiyonlar aşağıda özetlenmiştir:

4.2.2.1. Private Fonksiyonlar:

`UpdateModelInput()`	Model girdilerinin güncellenmesini sağlar.
`updateModelOutput()`	Model çıktılarının güncellenmesini sağlar.
`updateCARAMode()`	Yükseklikölçerin çalışma modu değişimlerine karar verme görevini gerçekleştirir.
`caraCycle()`	Yükseklikölçer modelinin normal çalışma modunda bir çevriminin çalıştırılmasını sağlar.
`warmupCycle()`	Yükseklikölçer modelinin warmup modunda bir çevriminin çalıştırılmasını sağlar.
`ibitCycle()`	Yükseklikölçer modelinin test modunda bir çevriminin çalıştırılmasını sağlar.

caraCalculate()	Yükseklikölçer modelinin ihtiyaç duyduğu hesaplamaların yapılmasını sağlar.
startWarmup()	Warmup moduna giriş için kullanılır.
endWarmup()	Warmup modundan çıkış için kullanılır.

4.2.2.2. Public Fonksiyonlar:

Run()	Yükseklikölçer model çevrimlerinin tek bir defa çalıştırılmaları için yazılmış model fonksiyonudur.
Init()	Modelin çalışma öncesi hazırlık işleri için kullanılacak model fonksiyonudur.
startIBIT()	Test moduna giriş için kullanılır. Test model dışından da talep edilebildiği için, bu fonksiyonlar public olarak hazırlanmıştır.
endIBIT()	Test modundan çıkış için kullanılır.

4.3.Taktik Hava Seyrüsefer Sistem Modeli

Taktik Hava Seyrüsefer unitesi genel olarak, uçak üzerinden seçimi gerçekleştirilen TACAN istasyonuna olan mesafe ve konum açısı bilgilerini hesaplamaktadır. Simulatör sistemlerinde gerçek TACAN istasyonları ile haberleşme mümkün olamayacağından bu istasyon bilgileri bir dosyada tutularak ve de uçağın pozisyon bilgisi ile Taktik hava seyrüsefer ünitesinden elde edilen mesafe ve konum açısı bilgilerini elde etmek mümkündür.

Buna göre, bu modelin çalışması için ihtiyacı olan girdi bilgileri ve bu bilgilerin edinildiği simulatör sistemleri şunlardır;

- Uçuş Modelinden elde edilen girdiler:
  - Uçağın anlık pozisyonu

- Kullanıcının yapmış olduğu girdiler:
  - Seçilen TACAN istasyon frekansı
  - Test işlemi

Simulatörün uçuş alanında bulunan tüm TACAN istasyon bilgileri (frekans, pozisyon, deniz seviyesinden yükseklik, morse kodu vb.) bir dosyada tutulmaktadır. Kokpitten içerisinden TACAN istasyonunun seçilmesive TACAN'ın aktif edilmesiyle model çalışmaya başlar ve sırasıyla şu işlemler gerçekleştirilir:

- Seçilen TACAN istasyonu ile ilgili bilgiler tüm TACAN istasyon bilgilerinin tutulu olduğu dosyadan bulunur.

- Uçağın anlık pozisyonu uçuş modelden elde edilir.

- İki pozisyon arasındaki mesafe ve konum açısı hesaplamaları gerçekleştirilir.

- Uçak ile istasyon arasındaki mesafe iletişim için uygun ise (istasyon uçağın alış mesafesi dışında ve görüşü engelleyecek konik alanın içinde olmamalıdır):
  - TACAN kontrol panelinden seçilmiş olan TACAN moduna göre görüntülenecek bilgiler HSI göstergesine gönderilir.
  - İstasyonun morse kodu kulaklıklardan yayınlanır.

Yukarıda da kısaca bahsedildiği gibi modelin çalışması sonucu elde edilen çıktılar şunlardır:

- İstasyona olan mesafe (feet)
- İstasyona olan konum açısı bilgisi
- İstasyon morse kodu

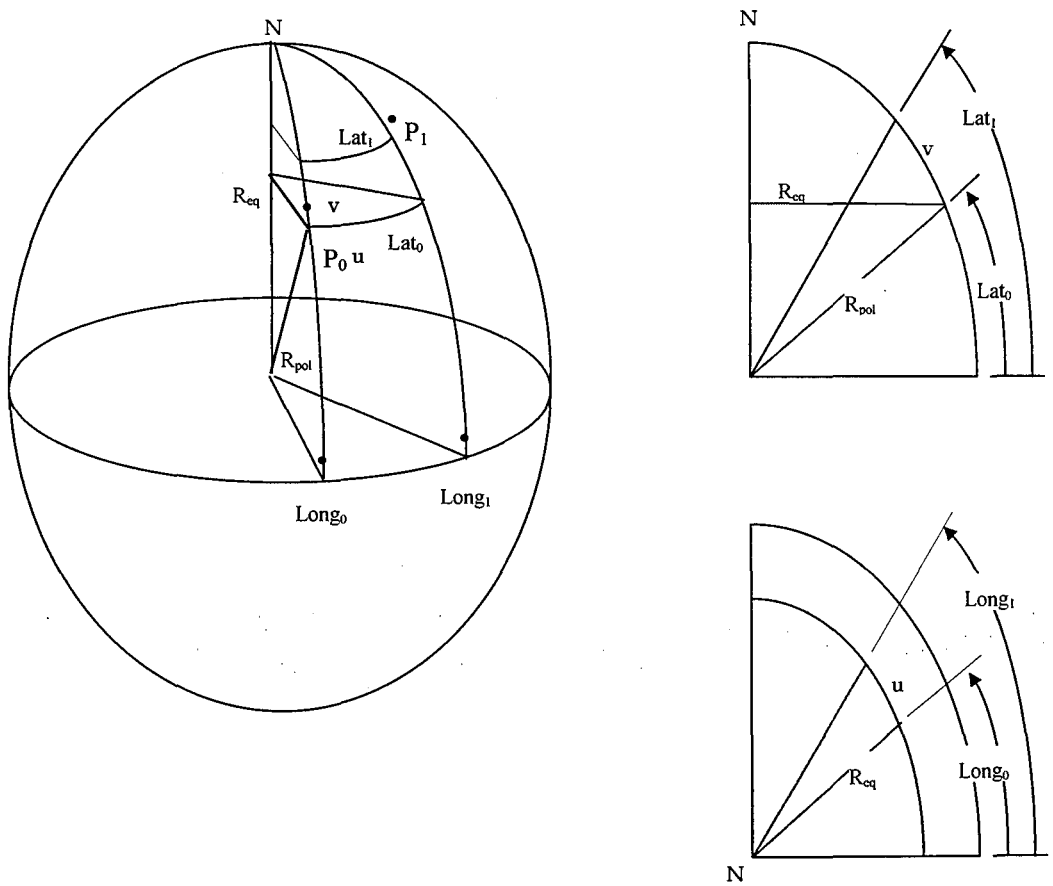
Her aviyonik ünitede olduğu gibi Taktik hava seyrüsefer ünitesinde de çalışmaya başlangıç sırasında ve kullanıcının isteği ile aktif ettiği test işlemi mevcuttur. Taktik hava seyrüsefer ünitesi test işlemi TACAN kontrol paneli üzerinden aktif edilmektedir. Bu her sistemde farklı yerden aktif edilebilmektedir.

Örneğin Taktik hava seyrüsefer ünitesi Mux Bus ile Aviyonik bilgisayara bağlı olsa idi, bilgisayarın okuyabildiği bir tuş takımından aktif edilmesi ve bilgisayarın bu test isteğini taktik hava seyrüsefer ünitesine bildirmesi gerekirdi. TACAN kontrol paneldeki test başlangıç butonu normal uçakta taktik hava seyrüsefer ünitesine seri olarak bağlanmış iken, simulatörde bağlantısı aynı kontrol panel modifiye edilip VME kartlar üzerinden kesikli sinyal ile gerçekleştirilmiştir. Bunun sebebi uygulama kolaylığının ihtiyacıdır.

4.3.1.TACAN matematik modeli:

4.3.1.1. Uçak ve seçilen istasyon arasındaki mesafe ölçümü

$P_0(lat_0, lon_0, alt_0)$ uçağın pozisyonu ve $P_1(lat_1, lon_1, alt_1)$ TACAN istasyonunun pozisyonu ise aradaki mesafe şu şekilde hesaplanmaktadır.



Şekil 4.6 Uçak ve seçilen istasyon arasındaki mesafe ölçümü

Enlem ve boylam bilgilerinin derece cinsinden radyan cinsine çevrilmesi gerekmektedir. Buna göre;

$$mesafe = \sqrt{u^2 + v^2 + (alt_1 - alt_0)^2} \quad (4\text{-}19)$$

$$Degree \times \pi / 180 = Radian \quad (4\text{-}20)$$

$$u = R_{eq} \times (long_1 - long_0) \quad (4\text{-}21)$$

$$v = R_{pol} \times (lat_1 - lat_0) \quad (4\text{-}22)$$

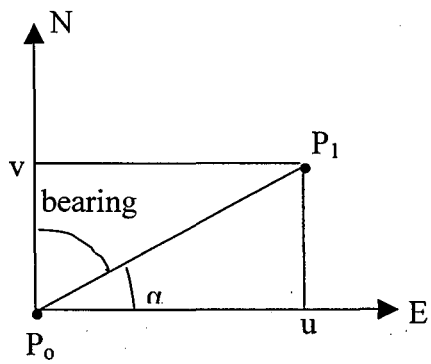
$$R_{eq} = R_{earth}(1 + ecc_{earth} \times \sin^2(lat_0))\cos(lat_0) \quad (4\text{-}23)$$

$$R_{pol} = R_{earth}(1 \quad 2ecc_{earth} + 3ecc_{earth} \times \sin^2(lat_0)) \quad (4\text{-}24)$$

$$Eccentrivcity\_of\_the\_earth = ecc_{earth} = 0{,}0033670039 \quad (4\text{-}25)$$

$$R_{earth} = 2{,}092647 \times 10^7 \; feet \quad (4\text{-}26)$$

4.3.1.2. Konum açısı bilgisinin hesaplanması



Şekil 4.7 Konum açısı hesaplaması

$$bearing = 90 - \alpha \quad (4\text{-}27)$$

$$\alpha = atn(v/u) \quad (4\text{-}28)$$

4.3.1.3. TO/FROM bilgisinin hesaplanması

$$heading - bearing \leq 90 \Rightarrow TO \quad (4\text{-}29)$$

$$heading - bearing > 90 \Rightarrow FROM \quad (4\text{-}30)$$

4.3.1.4. Yaklaşma yönünden sapma bilgisinin hesaplanması

$$course\_deviation = bearing - course \quad (4\text{-}31)$$

4.3.2.Taktik hava seyrüsefer yazılım modeli

C++ dilinde yazılmış olan modelin kaynak kodları Ek-3 de verilmiştir. Bu modelin sahip olduğu sınıflar ve bunların yapıları ana hatlarıyla aşağıda anlatılmaktadır. Buna göre;

```
#define C_OWNSHIP_DETECTION_RANGE 2500000
#define C_TACAN_BLIND_ZONE_ELEVATION 60
#define C_TCN_WARMUP_TIME 500
#define C_IBIT_TIME 240
#define C_TEST_0_TIME 200
#define C_TEST_1_TIME 100
#define C_TEST_2_TIME 0
// Equatorial Radius of the earth
#define    R_EARTH ((double) 2.092647e07 )
#define ECC_EARTH ((double) 0.0033670039 )
```

Tacan ünitesinin gerçek şartlarda bir alış mesafesi (Ownship Detection Range) ve TACAN istasyonunun belli bir mesafede iletiminin görülemediği kör bölge (TACAN Blind Zone Elevation) mevcuttur. Yine ünitenin normal çalışma şekline göre bir ısınma süresi (Warm up time) bulunmakta ayrıca istenilen zamanda ünite test (IBIT) edilebilmektedir. Test boyunca oluşan tepkileri modelde grçkleyebilmek için , Test fonksiyonu 3 kısıma (Test0, Test1, Test2) ayrılmıştır. Her bir bölümün harcadığı süre bellidir. Yine hesaplamalarda

kullanmak üzere dünyanın yarı çapları yukarıdaki gibi belirtilmiştir. Genel bir Taktik hava seyrüsefer modeli tasarımı olduğu ve kullanılan üniteye göre bu sürelerin farklılık göstermesi sebebi ve ayrıca yazılan modelin daha anlaşılır olması için gerekli süreler hazırlanmış olan modelde yukarıdaki şekilde sırasıyla alış mesafesi (C_OWNSHIP_DETECTION_RANGE), kör bölge (C_TACAN_BLIND_ZONE_ ELEVATION), ısınma (C_WARMUP_TIME) ve test süreleri (C_IBIT_ TIMER), (C_TEST_0_TIME), (C_TEST_1_TIME) (C_TEST_2_ TIME) ve dünyanin yari çaplari (R_EARTH) ve (ECC_EARTH) olarak tanımlanmıştır.

Taktik hava seyrüsefer modelinin çalışabilmesi için TACAN istasyon modellemesi de gerçekleştirilmelidir. Bu nedenle istasyon TACAN istasyonunun yayım yaptığı band tipi Band_Type tip tanımlaması ile yapılmıştır.

```
typedef enum {
band_X = 1,
        band_Y = 2
} Band_Type;
```

TACAN istasyonunun tipi Tacan_Type ile tanımlanmış, tipine özgün çalışma biçimi model içerisinde simüle edilimiştir.

```
typedef enum {
tacan_none = 0,
        tacan_air = 1,
        tacan_ground = 2
} Tacan_Type;
```

TACAN istasyonu ile uçak arasındaki iletişimi engelleyen bir arazi koşulu olup olmadığı `Occlusion_Type` ile tanimlanmiş ve model içerisinde simulasyonu gerçekleştirilmiştir.

```
typedef enum {
    occluded = 1,
    not_occluded = 2
} Occlusion_Type;
```

TACAN çalışma modlarının tanımlanması amacıyla aşağıda belirtilmiş olan `TacanMode_Type` tip tanımlaması yapılmıştır. TACAN çalışma modu değişkeni bu tip ile tanımlanmış, atamalar ve kullanım bu çerçevede içerisinde yapılmıştır.

```
typedef enum {
mode_OFF = 0,
      mode_REC = 1,
      mode_T_R = 2,
      mode_AA_REC = 3,
      mode_AA_T_R = 4
} TacanMode_Type;
```

TACAN istasyonuna göre durumun gösterildiği To/From tanımlaması `TacanToFrom_Type` ile gerçekleştirilmiştir.

```
typedef enum {
      tofrom_TO = 1,
    tofrom_FROM = 2
} TacanToFrom_Type;
```

TACAN istasyonu ile bağlantı sağlandığında pilotu uyarmak amacıyla bağlantı kurulduğuna dair intercom sistemine çıktı olarak gitmesi gereken morse kodu tipi `Callsign_Type` ile tanımlanmıştır. Bu tipte tanımlanmış olan her bir harf, 3 harfli istasyon kodunun her bir harfini oluşturmaktadır.

```
typedef struct {
        unsigned short letter1:5;
        unsigned short letter2:5;
        unsigned short letter3:5;
        unsigned short spare:1;
} Callsign_Type;
```

TACAN model hesaplamalarının gerçekleştirilmesi için uçağın pozisyon bilgisinin tanımlaması `Position_Type` Uçağın davranış bilgilerinin tanımlaması `Orientation_Type` ile gerçekleştirilmiştir. `Position_Type` tipinde tanımlanmış olan `latitude, longitude, altitude` bilgileri uçağın koordinat düzlemindeki pozisyon bilgisini oluşturmaktadır. `Orientation_Type` tipinde tanımlanmış olan `pitch, roll, heading` bilgileri uçağın kendi eksenine göre yaptığı davranışları oluşturmaktadır.

```
typedef struct {
        double latitude, longitude, altitude;
} Position_Type;
```

```
typedef struct {
        double pitch, roll, heading;
} Orientation_Type;
```

Koordinat eksenine göre tanımlanmış TACAN ile TACAN istasyonun pozisyonlarının x ve y koordinatlarına göre vektörel olarak ifade edilmesi amacıyla `Vector_Type` tanımlaması kullanılmaktadır.

```
typedef struct {
        double x, y;
} Vector_Type;
```

TACAN modelinin simule edilmesi için ihtiyaç duyduğu TACAN istasyonu ile ilgili girdiler için `Channel_Type` tanımlaması yapılmıştır.

```
typedef struct {
int no;
        Band_Type band;
        Callsign_Type callsign;
} Channel_Type;
```

`no`	TACAN istasyonunun sahip olabileceği kanal numaralarıdır. 1-126 arasında değer alabilmektedir..
`band`	TACAN X ve Y olmak üzere iki band üzerinden yayım yapabilmektedir.
`callsign`	Her bir TACAN istasyonunun sahip olduğu 3 harfli kod bulunmakta ve bağlantı kurulduğunda bu kodun iletimi gerçekleştirilmektedir.

Modelin simule edilebilmesi için uçak ile ilgili girdiler `Ownship_Type` tip tanımlaması ile yapılmıştır.

```
typedef struct {
TacanMode_Type tacanMode;
        Channel_Type channel;
        Position_Type position;
        Orientation_Type orientation;
} Ownship_Type;
```

tacanMode	TACAN çalışma modunu belirlemek için kullanılmaktadır. TACAN kontrol panelinden seçimi yapılan bu değere göre modelin çalışması değişiklik göstermektedir..
channel	TACAN kontrol panelinden seçilen TACAN kanal ve bandının tutulduğu değerdir. TACAN istasyonlarının sahip olabildiği kanal sayısı ile sınırlıdır.
position	Uçağın koordinat eksenine göre enlem, boylam ve yükseklik bilgilerinin tutulduğu değerdir. Bu bilgiler uçuş modelinden elde edilmektedir.
orientation	Uçağın kendi eksenine göre yunuslama, yalpa ve baş bilgisinin tutulduğu değerdir. Bu bilgiler uçuş modelinden elde edilmektedir.

Modelin simule edilebilmesi için istasyon ile ilgili girdiler TacanStation_Type tip tanımlaması ile yapılmıştır.

```
typedef struct {
        Tacan_Type type;
Channel_Type channel;
        Position_Type position;
```

```
        Occlusion_Type occlusion;
    } TacanStation_Type;
```

type	TACAN istasyonunun tipini belirlemektedir. Hava ve yer olmak üzere 2 tipi bulunmaktadır.
channel	TACAN istasyonunun kanal numarası tutulmakta ve kontrol paneli üzerinden seçilen değere göre hangi istasyon ile uçak arasında hesaplama yapılacağı belirlenmektedir. Kanal numarası ve bant değeri burada tutulmaktadır..
position	Seçilen kanal numarasına göre istasyonun pozisyon bilgileri buradan elde edilmekte ve matematik model hesaplamaları gerçekleştirilmektedir.
occlusion	Uçak ve istasyon pozisyon bilgileri elde edildikten sonra uçağın istasyon ile iletişimine engel yeryüzü engeli olup olmaması bilgisi görsel sistemden buraya bilgi olarak gelmektedir. Occlusion olması durumunda istasyon ile bağlantı sağlanmamaktadır.

Taktik hava seyrüsefer modelinin çalışması için ihtiyaç duyduğu girdiler için `TacanInput_Type` tip tanımlaması ve çalışma sonucu üretilen çıktılar için de `TacanOutput_Type` tip tanımlaması yapılmıştır.

```
    typedef struct {
    TacanMode_Type modeSwitch;
          int channelNo;
          Band_Type bandSwitch;
          bool ibitRequest;
          int course;
    } TacanInput_Type;
```

modeSwitch	Taktik hava seyrüsefer modelinin geçmesi istenilen mode değerini içerir. Mod değişiklik talepleri için kullanılır.
channelNo	TACAN kontrol panelinden seçilmiş olan kanal değerini tutmaktadır.
bandSwitch	TACAN kontrol panelinden seçilmiş olan band değerini tutmaktadır.
ibitRequest	Modelin test moduna geçmesi istenildiğinde kullanılır.
course	Yaklaşma yönü ve yaklaşma yönünden sapma hesaplamalarını gerçekleştirebilmek için gerekli bilgiyi tutmaktadır.

```
typedef struct {
int bearing;
      bool fRangeValid;
      int range;
      TacanToFrom_Type to_from;
      int courseDeviation;
      bool testLED;
      Callsign_Type callsign;
} TacanOutput_Type;
```

bearing	Uçağın istasyona olan bağıl baş bilgisidir.
fRangeValid	Uçak ile istasyon arasındaki mesafenin geçerli sınırlar içerisinde olup olmadığını tutan bayraktır. Çıktı bilgilerinin geçerli değer olup olmaması bu bayrağa göre belirlenmektedir.

range	Uçak ile istasyon arasındaki mesafe hesabıdır.
to_from	Uçağın istasyona doğru ya da istasyondan uzaklaşma bilgisidir.
courseDeviation	İstasyona seçilen yaklaşma ile yaklaşamama durumunda sapma açısıdır..
testLED	Test işlemi aktif edildiğinde, testin başladığını gösteren lambadır.
callsign	İstasyon ile iletişim kurulduğunda istasyonun iletilen morse kod bilgisidir.

Taktik hava seyrüsefer modelin işlenmesi sırasında girdiler ile çıktılar arasında ara hesaplamalar için ihtiyaç duyulan değerler için TacanInternal_Type tip yapılmıştır.

```
typedef struct {
        TacanMode_Type mode;
        TacanMode_Type prevMode;
        int warmupTimer;
        int ibitTimer;
        bool fBearingValid;
        bool fRangeValid;
        double deltaZ;
        double range;
        double bearing;
        ErrorCode_Type errorCode;
} TacanInternal_Type;
```

mode	TACAN çalışma modu tutulmaktadır.
prevMode	TACAN'ın bir önceki çalışma modu tutulmaktadır. TACAN ısınma moduna geçiş kontrolü için kullanılmaktadır.
warmupTimer	Modelin warmup zamanını modellemek için kullanılan zamanlayıcı değişkenidir.
ibitTimer	Modelin test modunda kaldığı zamanı modellemek için kullanılan zamanlayıcı değişkenidir.
fBearingValid	Uçağın seçilen moduna göre ya da istasyonun alış alanı içerisinde olmaması durumunda bayrak aktif olmaktadır.
fRangeValid	İstasyonun alış alanı içerisinde olmaması durumunda aktif olan bayraktır.
deltaZ	Uçağın istasyonu göremediği kör konik alanı içerisinde olup olmadığının hesabı yapılırken kullanılan bir değerdir.
range	Uçak ile istasyon arasındaki mesafedir.
bearing	Uçağın istasyona olan bağıl baş bilgisi değeridir.

Modelin çalışması için ihtiyaç duyulmakla beraber girdi ya da çıktı olmayan diğer değişkenler, model sınıfının *private* değişkenler bölümünde tanımlanmıştır. Model sınıfı C_TACAN_Model aşağıda verilmiştir.

```
class C_TACAN_Model {
      private:
            TacanInput_Type modelInput;
            TacanOutput_Type modelOutput;
            TacanInternal_Type locals;
```

```
                Ownship_Type ownship;
                TacanStation_Type lockedStation;
                bool findMatchingStation();
                bool isStatonInOwnshipDetectionRange();
                bool isInStationBlindZone();
                double calculateBearing();
                int calculateDeviation();
                double calculateRange();
                TacanToFrom_Type calculateToFrom();
                int invalidBearingValue();
                void updateModelOutput();
                void processInputs();
                void processModeSwitch();
                void processIbitRequest();
                void ibitCycle();
                void tacanCalculate();
                Vector_Type  &deltaPPCalc(  Vector_Type
pos0, Vector_Type pos1 );
        public:
                C_TACAN_Model();
                ~C_TACAN_Model();
                void Run();
                void Init();
                void updateModelInput();
                void requestIBIT();
                Position_Type
&GetStationPosition(unsigned int no);
                void  SetOwnshipPosition(  Position_Type
pos ); };
```

Taktik hava seyrüseferinin modellenmesi için kullanılan *private* ve *public* fonksiyonlar aşağıda özetlenmiştir:

4.3.2.1. Private Fonksiyonlar:

`modelInput`	Model girdilerinin güncel hale gelmesini sağlar. `TacanInput_Type` tip tanımlamasında bu tipten bahsedilmiştir.
`modelOutput`	Model çıktılarının güncellenmesini sağlar. `TacanOutput_Type` tip tanımlamasında bu tipten bahsedilmiştir.
`locals`	Modelde çıktı olarak kullanılmayacak değerlerin tutulduğu yapıdır. `TacanInternal_Type` tip tanımlamasında bu tipten bahsedilmiştir.
`ownship`	Uçağın pozisyon davranış gibi sahip olduğu bilgileri tutmaktadır. `Ownship_Type` tip tanımlamasında bahsedilmiştir.
`lockedStation`	Bağlantı kurulmuş olan istasyonunun sahip olduğu bilgileri tutmaktadır. `TacanStation_Type` tip tanımlamasında bu tipten

	bahsedilmiştir.
findMatchingStation()	Uçak üzerinden seçilmiş olan kanal ve banda uygun TACAN istasyonu olup olmadığını hesaplayan fonksiyondur..
isStationInOwnshipDetectionRange()	Bulunan istasyonun alış mesafesi içinde olup olmadığını hesaplayan fonksiyondur.
isInStationBlindZone()	Uçağın kör konik alan iöerisinde olup olmadığını hesaplayan fonksiyondur.
calculateBearing()	İstasyon ile uçak arasındaki bağıl baş bilgisini hesaplayan fonksiyondur.
calculateDeviation()	Seçilen yaklaşma açısına göre mevcut sapma açısını hesaplayan fonksiyondur.
calculateRange()	Uçak ile seçili takan istasyonu arasındaki olan mesafeyi ölçmek için kullanılan fonksiyondur.
calculateToFrom()	Uçağın istasyona doğrumu gidiş ya da istasyondan uzaklaşışı hesaplayan fonksiyondur.
invalidBearingValue()	Bağıl baş bilgisinin geçerli olmadığı durumlarda konum açısı göstergecinin

	davranışını gerçekleyen fonsiyondur.
`updateModelOutput()`	Modelde yapılan hesaplamalar sonucu her bir model döngüsü sonunda çıktı bilgilerini güncelleyen fonksiyondur.
`processInputs()`	Girdi bilgilerinin elde edildiği fonksiyondur.
`processModeSwitch()`	Mode değerine göre modelin içerisine girmeden ısınma sürecinin başlatıldığı ya da ısınma sürecinin atlanarak modelin çalıştırıldığı fonksiyondur.
`processIbitRequest()`	Modelin test moduna geçiş durmunu hazırlayan fonksiyondur.
`ibitCycle()`	Modelin test modunda bir çevriminin çalıştırılmasını sağlayan fonksiyondur.
`tacanCalculate()`	Taktik hava seyrüsefer modelinin bir çevrim çalışmasını sağlayan fonksiyondur.

4.3.2.2. Public Fonksiyonlar:

`Init()`	Modelin çalışma öncesi hazırlık işleri için kullanılacak model fonksiyonudur.

`Run()`	Model çevrimlerinin tek bir defa çalıştırılmaları için yazılmış model fonksiyonudur.
`updateModelInput()`	Modele yapılan girdilerin güncellemesini sağlayan fonksiyondur.
`requestIBIT()`	Test moduna giriş için kullanılır. test model dışından da talep edilebildiği için, bu fonksiyonlar public olarak hazırlanmıştır.
`GetStationPosition()`	Seçili istasyonun poziisyon bilgisini döndüren fonksiyondur.
`SetOwnshipPosition()`	Uçuş modelinden elde edilen uçağın pozisyon ve durum bilgilerini döndüren fonksiyondur.

4.4.Dost/düşman Tanıma Sistem Modeli

Dost/düşman tanıma sistem modeli, radyo modelinde olduğu gibi IFF kontrol panel üzerinden yapılmakta olan anahtar değişikliklerine göre fonksiyonlarını gerçekleştirmektedir.

Dost/düşman tanıma sistemi ile ilgili sorgulamalar eğitmen konsolu üzerinden eğitmen tarafından aktif edilmekte ve buna göre öğrenci pilota Dost/düşman tanıma sorgulamalarıyla ilgili alışkanlıklar kazandırılmaktadır.

Önceden de bahsedilmiş olan Dost/düşman tanıma modlarının aktivasyonu IFF kontrol panel üzerinden gerçekleştirilmektedir. VME kartlar üzerinden digital olarak her bir mod için bir hat kullanılarak gerçekleştirilmektedir. Ana bilgisayar tarafından okunan Dost/düşman tanıma mod aktivasyon durumları eğitmen konsol üzerinden takip edilebilmektedir.

Mod 1 ve Mod 3/A kodlamaları da bu panel üzerinden gerçekleştirilmektedir. Mod aktivasyon ve mod değerleri VME kartlar ile ana bilgisayara okunmaktadır.

Mod 1 değeri 00 ile 73 arasında değer almakta, Onlar basamağı maksimum 7 değerini birler basamağı maksimum 3 değerini almaktadır. Her bir basamağın

oluşturulması için kontrol panel üzerinden okunan 5 ikilik bit değeri kullanılmaktadır. Bu durumda birler basamağı için 2, onlar basamağı için 3 kesikli sinyal okuması gerçekleştirilmektedir.

Mod 1 değeri için kontrol panelde çıktı değeri şu şekilde kodlanacak olursa;

Çizelge 4.6 Mod 1 Çıktı Değer Kodlaması

Onlar			Birler	
4	2	1	2	1
A	B	C	D	E

$$Mod1 = (10 \times ((4 \times A) + (2 \times B) + (1 \times C))) + (2 \times D) + (E) \qquad (4\text{-}32)$$

Mod 3/A değeri 0000 ile 7777 arasında değer almakta, Tüm basamaklar maksimum 7 değerini almaktadır. Her bir basamak binary olarak kontrol panel üzerinden okunmaktadır. Bu durumda her bir basamak için 3 kesikli sinyal okuması gerçekleştirlmektedir.

Mod 3/A değeri için kontrol panelde çıktı değeri şu şekilde kodlanacak olursa;

Çizelge 4.7 Mod 3/A Çıktı Değer Kodlaması

Binler			Yüzler			Onlar			Birler		
4	2	1	4	2	1	4	2	1	4	2	1
A	B	C	D	E	F	G	H	I	J	K	L

$$Mod3/A = (1000 \times ((4 \times A) + (2 \times B) + (1 \times C))) + (100 \times ((4 \times D) + (2 \times E) + (1 \times F))) +$$

$$(10 \times ((4 \times G) + (2 \times H) + (1 \times I))) + (((4 \times J) + (2 \times K) + (1 \times L))) \quad (4\text{-}33)$$

Yukarıdaki şekilde hesaplanan değerler eğitmen konsolda eğitmene görüntülenmek üzere gönderilmektedir.

Mod 4 için Dost/düşman tanıma sistemi haricinde özel bir ünite bulunmaktadır. Bu modun sakladığı değerler gizlidir ve mod yükleme işlemi bakımcılar tarafından gerçekleştirilmektedir. Sorgulamalar sırasında Mod 4 bilgisinin yüklü olup olmaması yapılan sorgulamalara verilen tepki açısından önemlidir. Burada amaç mod 4 bilgisinin nasıl yüklendiği değil, yüklü olup olmamasıdır. Bu nedenle eğitmen konsoluna Mod 4 bilgisinin yüklü olma ve olmama durumunun seçilebilmesi için bir buton konulmakta ve eğitmen tarafından bunun durumuna karar verilmektedir.

Sorgulama işlemi eğitmen konsolundan gerçekleştirilmektedir. İki tür sorgulama yapılabilmektedir Bunlar, dost ve düşman sorgulaması. Her iki uçağın da Mod 4 bilgileri aynı ise yapılan sorgulama dost olmaktadır. Ancak bu sistemde iki uçak olmadığından diğer uçak varmış gibi dost düşman seçimi eğitmen konsolundan gerçekleştirilmektedir.

Sorgulama yapılsa da yapılmasa da Mod 4 bilgisi yüklü değil ise kokpitte bulunan IFF uyarı lambası sürekli yanmaktadır. Yine bu lambanın kontrolü VME kartlar üzerinden gerçekleştirilmektedir. Bunun dışında kod yüklü ise ve dost sorgulaması gerçekleşiyorsa IFF kontrol paneli üzerindeki cevaplama yapıldığını gösteren lamba yanıp sönmekte, IFF sesi aktif edilmiş ise öğrenci kulaklığından, cevaplama sırasında ilgili sinyal sesi duyulmaktadır. Yapılan sorgulama düşman sorgulaması ise sadece Dost/düşman tanıma uyarı lambası yanıp sönmekte bu şekilde pilotu bilgilendirmektedir.

4.4.1.Dost/düşman tanıma sistemi yazılım modeli

Aşağıda C++ dilinde yazılmış olan modelin sahip olduğu tanımlamalar ve bunların yapıları ana hatlarıyla anlatılmaktadır. Bu modelde çalıştırılması gereken bir matematik model bulunmamaktadır. Kokpit içerisindeki anahtar konumlarına

göre ve eğitmen konsolundan verilen sorgulama çeşidine göre bir tepki oluşmaktadır. Buna göre anahtarların okuma tanımlamaları yapılmış ve çalışma mantığına göre bir matriks oluşturulmuştur. Buna göre bu modelde yapılmış olan tanımlamalar şu şekildedir;

Yazılım modeli içerisinde Dost/düşman tanıma sistemi çalışma modlarının tanımlanması amacıyla aşağıda belirtilmiş olan IFFMode_Type tip tanımlaması yapılmıştır. Dost/düşman tanıma sistem çalışma modu değişkeni bu tip ile tanımlanmış, atamalar ve kullanım bu çerçevede içerisinde yapılmıştır. Dost/düşman tanıma sisteminin her bir çalışma moduna özgün davranış biçimi yazılım modeli içerisinde de simüle edilmiştir.

```
typedef enum {
        Mast_Sw_OFF  = 1,
        Mast_Sw_Stby = 2,
        Mast_Sw_Low  = 3,
        Mast_Sw_Norm = 4,
        Mast_Sw_Emer = 5,
} IFFMode_Type //Master_Sw_EnumType;
```

Dost/düşman tanıma sisteminin sahip olduğu modları kontrol eden anahtarların yazılım modelinde kullanılması için aşağıda belirtilen Mode1_Sw_EnumType, Mode2_Sw_EnumType, Mode3A_Sw_Enum Type, ModeC_Sw_EnumType, Mode4_Sw_EnumType, Mode4_ Audio_Sw_EnumType tip tanımlamaları gerçekleştirilmiştir. Bu tanımlamalardaki değerler mod anahtarlarının fiziksel konumlarına karşılık gelmektedir. Modelin sahip olduğu çalışma moduna göre modlar değrelendirilmektedir.

```
typedef enum {
        Mode1_Sw_OUT    = 1,
        Mode1_Sw_ON     = 2,
} Mode1_Sw_EnumType;
```

```
typedef enum {
        Mode2_Sw_OUT  = 1,
        Mode2_Sw_ON   = 2,
        Mode2_Sw_TEST = 3,
} Mode2_Sw_EnumType;
```

```
typedef enum {
        Mode3A_Sw_OUT  = 1,
        Mode3A_Sw_ON   = 2,
        Mode3A_Sw_TEST = 3,
} Mode3A_Sw_EnumType;
```

```
typedef enum {
        ModeC_Sw_OUT  = 1,
        ModeC_Sw_ON   = 2,
} ModeC_Sw_EnumType;
```

```
typedef enum {
        Mode4_Sw_OUT    = 1,
        Mode4_Sw_ON     = 2,
} Mode4_Sw_EnumType;
```

```
typedef enum {
        Mode4_Audio_Sw_OUT     = 1,
        Mode4_Audio_Sw_AUDIO   = 2,
        Mode4_Audio_Sw_LIGHT   = 3,
} Mode4_Audio_Sw_EnumType;
```

```
typedef enum {
        ModeValue_High  = 1,
        ModeValue_Low   = 2,
} ModeValue_EnumType;
```

IFF kontrol paneli üzerinde kimlik özelliğini kontrol etmek amacıyla bulunan anahtarının yazılım modelinde kullanılabilmesi için aşağıda belirtilen `IP_Sw_EnumType` tip tanımlaması yapılmıştır.

```
typedef enum {
        IP_Sw_OUT     = 1,
        IP_Sw_IDENT   = 2,
} IP_Sw_EnumType;
```

Eğitmen konsolu üzerinden kontrol edilen Reply butonu, modelin çalışmasını sağlayan butondur. Reply butonuna basılmasıyla model bir döngü çalışmaktadır. Yazılım modelinin aktif olmasını sağlayan Reply butonunun tip tanımlaması `Reply_Mode_EnumType` ile gerçekleştirilmiştir.

```
typedef enum {
        Reply_Mode_FOE      = 1,
        Reply_Mode_FRIEND   = 2,
} Reply_Mode_EnumType;
```

Eğitmen konsolundan seçimi gerçekleştirilen Zeroize butonunun tip tanımlaması `Zeroize_EnumType` ile gerçekleştirilmektedir.

```
typedef enum {
        Zeroize_Selected          = 1,
        Zeroize_NOT_Selected      = 2,
} Zeroize_EnumType;
```

IFF lambası, Reply lambası, test lambası ve Master Caution lambalarının kontrolü için `Light_EnumType` tip tanımlaması gerçekleştirilmiştir.

```
typedef enum {
        Light_ON        = 1,
        Light_OFF       = 2,
} Light_EnumType;
```

Ses sistemine gönderilecek olan bilginin tip tanımlaması `VOICE_EnumType` ile gerçekleştirilmiştir.

```
typedef enum {
        VOICE_ON        = 1,
        VOICE_OFF       = 2,
} VOICE_EnumType;
```

Dost/düşman tanıma sistem modelinin çalışması için ihtiyaç duyduğu girdiler için `IFFInput` tip tanımlaması ve çalışma sonucu üretilen çıktılar için de `IFFOutput` tip tanımlaması yapılmıştır.

```
typedef struct {
        Master_Sw_EnumType Mast_Sw ;
        Mode1_Sw_EnumType  Mode1_Sw ;
        Mode2_Sw_EnumType Mode2_Sw ;
        Mode3A_Sw_EnumType Mode_3A_Sw ;
        Mode4_Sw_EnumType Mode4_Sw  ;
        ModeC_Sw_EnumType Mode_C_Sw     ;
        Mode4_Audio_Sw_EnumType Mode4_Sw_Audio ;
        IP_Sw_EnumType IP_ident_Sw    ;
        Reply_Mode_EnumType Reply_Mode ;
        Zeroize_EnumType Zeroize_Pos ;
        ModeValue_EnumType Mode1_Ones_1;
        ModeValue_EnumType Mode1_Ones_2;
        ModeValue_EnumType Mode1_Tens_1;
        ModeValue_EnumType Mode1_Tens_2;
        ModeValue_EnumType Mode1_Tens_4;
        ModeValue_EnumType Mode3A_Ones_1;
        ModeValue_EnumType Mode3A_Ones_2;
        ModeValue_EnumType Mode3A_Ones_4;
        ModeValue_EnumType Mode3A_Tens_1;
        ModeValue_EnumType Mode3A_Tens_2;
        ModeValue_EnumType Mode3A_Tens_4;
        ModeValue_EnumType Mode3A_Hundreds_1;
        ModeValue_EnumType Mode3A_Hundreds_2;
```

```
        ModeValue_EnumType Mode3A_Hundreds_4;
        ModeValue_EnumType Mode3A_Thousands_1;
        ModeValue_EnumType Mode3A_Thousands_2;
        ModeValue_EnumType Mode3A_Thousands_4;
    }IFFInput;
```

Girdi tipini oluşturan elemanların detaylı açıklamaları için Dost/düşman tanıma sistemi modelinin teorik açıklamasına bakılabilir. Girdi olarak verilen tüm tipler birer anahtardır ve bu yapıda bu anahtarların okuma işlemleri gerçekleştirilmiştir.

```
    typedef struct {
        Light_EnumType Reply_lt;
        Light_EnumType Master_Cation_lt;
        Light_EnumType IFF_lt;
        Light_EnumType Ident_lt;
        int Mode1   ;   // Mode 1  Value
        int Mode_3A ;   // Mode 3 A Value
        VOICE_EnumType IFF_Wrn_Voice_Mess;
    }IFFOutput;
```

Çıktı tipini oluşturan elemanların kısa açıklamaları aşağıda verilmiştir.

`Reply_lt`	Modelin çalışması sonucu şartlar uygun olduğunda lambanın aktivasyonu gerçekleştirilir.
`Master_Cation_lt`	Modelin çalışması sonucu şartlar uygun

	olduğunda lambanın aktivasyonu gerçekleştirilir.
IFF_lt	Modelin çalışması sonucu şartalr uygun olduğunda lambanın aktivasyonu gerçekleştirilir.
Ident_lt	Kokpit içerisinden kimlik anahtarı aktif edildiğinde bu lamba aktif edilir.
Model	Hesaplanan Mod 1 değeri eğitmen konsolu ekranına gösterilmek üzere gönderilir..
Mode_3A	Hesaplanan Mod 3A değeri eğitmen konsolu ekranına gösterilmek üzere gönderilir

Modelin çalışması için ihtiyaç duyulmakla beraber girdi ya da çıktı olmayan değişkenler, model sınıfının *private* değişkenler bölümünde tanımlanmıştır. Model sınıfı `C_IFF_Model` aşağıda verilmiştir.

```
class C_IFF_Model {

    private:
        void updateModelInput();
        void updateModelOutput();
        void updateIFFMode();
        void IFFCycle();
        void IFFModeValueCalculate();
    public:
        IFFInput modelInput;
        IFFOutput modelOutput;
        C_CARA_Model();
```

```
            ~C_CARA_Model();
            void Init();
            void Run();
};
```

Yükseklikölçerin modellenmesi için kullanılan *private* ve *public* fonksiyonlar aşağıda özetlenmiştir:

4.4.1.1. Private Fonksiyonlar:

`UpdateModelInput()`	Model girdilerinin güncellenmesini sağlar.
`updateModelOutput()`	Model çıktılarının güncellenmesini sağlar.
`updateIFFMode()`	IFF çalışma modu değişimlerine karar verme görevini gerçekleştirir.
`IFFCycle()`	Dost/düşman tanıma sistem modelinin normal çalışma modunda bir çevriminin çalıştırılmasını sağlar.
`IFFModeValueCalculate()`	Dost/düşman tanıma sistem modelinde bulunan Mod 1 ve Mod 3A değerlerinin hesaplandığı kısımdır.

4.4.1.2. Public Fonksiyonlar:

`Run()`	Dost/düşman tanıma sistem model çevrimlerinin tek bir defa çalıştırılmaları için yazılmış model fonksiyonudur.
`Init()`	Modelin çalışma öncesi hazırlık işleri için kullanılacak model fonksiyonudur.

5. SONUÇLAR VE TARTIŞMA

Bu tezde; hava veri bilgisayarı, yükseklikölçer, taktik seyrüsefer ve dost/düşman tanıma sistemi gibi aviyonik ünitelerin simülatör sistemleri için modellemeleri incelenmiştir. Bu ünitelerin modellerine ihtiyaç duyulmasının sebebi hepsinin spesifik çalışma şartlarının bulunması ve bu şartların simulatör sistemlerine uygun yapıda olmamasıdır.

Bu tür aviyonik ünitelerin modellenmesine tek ihtiyaç duyulan yer simulatör sistemleri değildir. Aynı zamanda, bu ünitelerin entegre edilecekleri sistemlerin laboratuarlarında da entegre edildiği ana sistemin bütününü test ederken kullanılmasına ihtiyaç duyulmaktadır. Bunun sebebi, test ortamlarında da bu ünitelerin çalışma şartları gerçeklenemediği ve bu ünitenin sistemin diğer elemanlarına gönderdiği çıktı bilgilerine ihtiyaç duyulmasıdır.

Genel simulatör konseptine uygun olarak her sistemin çalışma prensibi belirlidir. Üzerinde çalışılan ünitelerin modelleri, mevcut simulatör ihtiyaçlarını karşılayacak düzeyde gerçekleştirilmiş, diğer sistemlerin halihazırda hesapladığı değerler bulunmakta ise bunlar ilgili sistemden girdi olarak elde edilmiş ve bu girdilere göre modelden beklenen çıktılar hesaplanmıştır. Böyle sistemlerde gerçek ünitenin ihtiyacı olan girdilerin modellerinde de girdi olması beklenmemelidir. Önemli olan çıktıların doğru bir şekilde elde edilebilmesi ve sistem işleyinin beklenen doğrultuda olmasıdır.

Bunun bir örneği hava veri bilgisayar ünitesinin modellemesi sırasında ortaya kondu. Gerçek şartlarda sensörlerden elde ettiği girdi bilgileri yerine bu ünitenin modelinde, sistemde halihazırda bulunan gerçek hava hızı ve deniz seviyesinden yükseklik değerleri girdi bilgisi olarak kullanıldı. Gerçek ünitede, bu iki değer, uçağı çevreleyen sensörlerden alınan havanın değerleri ideal gaz denklemleri ile hesaplanarak çıktı bilgisi olarak elde edilirken, modelde hem girdi hem de çıktı bilgisi olmuştur. Tüm simülasyon boyunca sensörlerin verdiği hava şartları ile ilgili değerler doğru olarak sisteme beslenemeyeceği için girdi bilgisi olarak simulatör sistemindeki uçuş modelinden gerçek hava hızı, görsel sistemden de deniz seviyesinden yükseklik elde edildi. Bunlar ideal gaz denklemlerinde tersinir işleme sokularak diğer çıktı bilgilerini hesaplamak için gerekli olan

değerleri hesaplamakta kullanıldı. Böylelikle sistemin daha doğru bir şekilde çalışması sağlanmış oldu.

Bu tezde bahsedilmemesine rağman bir diğer örnek de ataletsel seyrüsefer sistemidir. Simulatör sisteminde, bu ünitenin üretmiş olduğu pozisyon ve hız bilgilerini, kokpit içerisindeki uçuş kontrol elemanlarından aldığı mekanik tepkilere göre uçuş modeli üretmektedir. Normal şartlarda uçak üzerinde ivmeölçerler ile üretilmiş olan bu bilgi, simulatör sistemlerinde uçuş modelinde üretilmiş olduğundan, EGI ünitesinin bu hesaplamayı yapmasını lüzumsuz kılmıştır. Zaten simulatör sisteminde, ivme ölçerlerin çalışması da mümkün değildir. Bu nedenle EGI ünitesinin modellenmesinde girdi olan pozisyon bilgisi yine modelin çıktısı olmaktadır. Bu tezde, EGI ünitesinin de modellenmesinin incelenmesi düşünülmüş ancak bu modelin içerdiği genel modelleme ile ilgili özellikler diğer aviyonik ünitelerce incelendiğinden ve uçuş modelinde EGI ünitesinin tüm hesaplamaları gerçekleştirildiğinden bu modelin incelenmesinden vazgeçilmiştir.

Bir modelde gerçekleştirilmiş bir hesabın, bir ikinci modelde tekrar yapılması, hem yapılan işin tekrar edilmesi hem de sistemler arasındaki koordinasyonun kaybedilmesine sebep olur. Hesaplamaların dublikasyonu sebebiyle olabilecek farklılıklar, sistemlerin uyumsuz çalışmalarına sebep olabilmektedir. Taktik hava seyrüsefer modelinde, TACAN istasyonu ile uçak arasındaki mesafe ve konum açısı gibi bilgileri hesaplarken, uçağın pozisyon bilgisine ihtiyaç duyulmakta ve bu bilgi uçuş modelinden elde edilmektedir. Eğer bu model dahilinde pozisyon hesabı tekrar yapılıyor olsa idi, iki hesap arasındaki en ufak fark taktik hava seyrüsefer model çıktılarının yanlış sonuç oluşturmasına sebep olabilecek ve yatay durum göstergesine gönderilen konum açısı ve mesafe bilgisi yanlış değere sahip olabilecekti. Çok küçük olarak gözüken bu yanlış değer, uzak mesafelerde çok farkedilmeyecek ancak yakın mesafelerde kendini gösterecektir. Görsel sistemde TACAN istasyonunun göründüğü yer ile yatay durum göstergesinin gösterdiği değer farklı olacaktır. Bu nedenle tek kaynaklı bir bilgi dolaşımı böyle çoklu sistemlerde oldukça önemlidir.

Özellikle radyo dalgalarının kullanıldığı sistemlerin modellenmelerinde gerçek şartlardan çok daha ideal sonuçlar ortaya çıkmaktadır. Böyle durumlarda kullanıcının verdiği bilgiler doğrultusunda gerçeğe yakın olacak şekilde modele girdiler yapılabilmekte, sistemin işleyişi gerçeğe uygun hale getirilmeye çalışılmaktadır.

Yükseklikölçer modeli gerçekleştirildiğinde uçağın her durumunda yer seviyesinden yüksekliği ölçebilmekteydi. Kullanıcının her zaman bu bilgiyi sağlıklı bir şekilde alamadıklarını söylediklerinde yapılan incelemeler sonunda bu üniteye ait bir tek anten olduğu ve bu sebeple uçağın yapmış olduğu manevraların antenin yer yüzeyini engellediği ortaya çıktı ve buna göre bir çalışma modeli oluşturuldu. Teknik kaynaklı olmayan bunun gibi bir çok konu da modelin istenilen şekilde çalışmasını engellemektedir. Bu nedenle modelinin gerçekleştirildiği sistemlerin kullanıcılarıyla iletişim içinde olmak oldukça önemlidir.

Simulatör sistemlerinde genellikle C++, ya da daha az sıklıkla ADA dili kullanılmaktadır. ADA dilinin tercih edildiği durumlarda en önemli amaç bu dilin yazılımcının hata yapmasına izin vermemesidir. Ancak çok yaygın olarak kullanılan bir dil olmaması sebebiyle genellikle C++ dili bu tür modellerin hazırlanmasında tercih edilmektedir. C++ dilinin esnek bir dil olması her türlü uygulamayı gerçekleştirebiliyor olmak, nesne tabanlı olması C++ dilinin kullanılmasında tercih sebebi olmaktadır. C++ dilinin en önemli dezavantajı güvenlik açıklarının bulunmasıdır. Yapılan projenin önemine ve gizlilik derecesine göre bu dilin kullanılması düşünülmelidir.

Bu çalışma boyunca, sistemlerin modellerinin hazırlanması sırasında dikkat edilmesi gereken:

- modellenen sistemin her koşuldaki işleyişinin anlaşılması,
- sistemin diğer sistemlerle olan arayüzüne göre modelin hazırlanması
- veri akış koordinasyonunun önemi incelenmiştir.

Yapılan bu çalışma ile simulatör sistemlerinde modelleme ile ilgili çalışmalara rehber olabilecek bir başlangıç, konu üzerinde çalışanlara başvurulabilecek bir materyal olabilmesi amaçlanmıştır.

KAYNAKLAR

1.*Specification for the Central Air data Computer (CADC CDRL No.002)*, Publication No. 107010 Rev A, Austronautics C. A. Ltd., U.S.A.

2. KAYTON, M. ve FRIED W. R., *Avionics Navigation Systems*, A Wiley Interscience Publication, John Wiley & Sons, Inc.,2nd Edition, U.S.A (1997).

3.*System Engineering Course Document*, Israel Aircraft Industry (IAI), MHT (Technical Publications and Training) (1999).

4. KARAMANCIOĞLU A., *Short Term Dynamic Behaviour Improvement of A light Transport Aircraft*, Osmangazi University, Eskişehir, Second Edition, December (1993).

5. GREEVY J. M., *LN-100G EGI Senior Engineering Instructors*, Litton Technical Training, Litton Company (1999).

6. KELLY A, *Introduction to Mobile Robots, Position Estimation 4: Inertial Navigation System*, Carnege Mellon University, Fall (1996).

7. POWELL J., *Aircraft Radio Systems*, BAC Engineering, GradMA.

8. *Federal Radio Navigation Plan,*U.S. Departments of Defense and Transportation. DOT-VNTSC-RSPA-95-1/DOD-4650.5. Washington, DC (1994).

9. *Climatic Extremes for Military Equipment,* MIL-STD 210A.

10.*IFF Functional Requirement specification (FRS) Document*, Israel Aircraft Industry Lahav Division, Document Identifier: TAU543/990979, (2000).

11.http-1: http://www.enocta.com/tr/kaynaklar_makale_detay.asp?url=91.

12.*Simulator Equipment Training Course*, Environmental Tectonics Corporation, Aeromedical Training Institude, Southampton, U.S.A (1990).

13.STROUSTRUP, B., *The C++ programming language* Addison-Wesley, (1997).

14.COESA, *é1962 U.S. Standart Atmosphere Tables,* Standart Atmosphere Commity, U.S.A (1962).

15. GALLAGHER G. L., HIGGINS L. B., KHINOO L. A., ve PIERCE P. W., *Fixed Wing Performance / Flight Test Manual*, Naval Test Pilot School USNTPS-FTM-No. 108, U.S.A (1992).

16.*Military Specifications MIL-STD-1553B: Digital Time Division Command/Response Multiplex Data Bus, USAF, U.S.A (1996).*

17. DUNN, A.M, *MIL STD 1553 Tutorial*, Engineering Manager Bus Analyser Product Development FairChild Communications & Electronics Company (1992)

18.*MIL-STD-1553 Designer's Guide*, DDC ILC Data Device Corporation, 3rd Edition, (1990).

EKLER

Bu bölümde okuyucuya referans olması amacıyla, modellemesi gerçekleştirilmiş aviyonik ünitelerden Hava veri bilgisayar modeli, taktik hava seyrüsefer modeli ve yüksekliköl̈çer modelinin kaynak kodları verilmiştir.

EK-1 Hava Veri Bilgisayar Modeli Kaynak Kodu

```
#include <math.h>
#include "main.h"
#include "adcModel.h"

Atmos_data     gr_ADC_model_AtmosData;
float FrameTime;
static int test_flag = TRUE;
static double dLastValidValueBaroPresSet = ADC_BARO_SET_NOM;

 /* altitude breakpoints for atmospheres (ft)  */

float atmos_alt[N_ATM_ALT] = {
     0000.,   1000.,   2000.,   3000.,   4000.,   5000.,   6000.,
   7000.,  8000.,  9000.,
    10000.,  11000.,  12000.,  13000.,  14000.,  15000.,  16000.,
   17000., 18000., 19000.,
    20000.,  21000.,  22000.,  23000.,  24000.,  25000.,  26000.,
   27000., 28000., 29000.,
    30000.,  31000.,  32000.,  33000.,  34000.,  35000.,  36000.,
   37000., 38000., 39000.,
    40000.,  41000.,  42000.,  43000.,  44000.,  45000.,  46000.,
   47000., 48000., 49000.,
    50000.,  51000.,  52000.,  53000.,  54000.,  55000.,  56000.,
   57000., 58000., 59000.,
    60000.,  61000.,  62000.,  63000.,  64000.,  65000.,  66000.,
   67000., 68000., 69000.,
    70000.,  71000.,  72000.,  73000.,  74000.,  75000.,  76000.,
   77000., 78000., 79000.,
    80000.,  81000.,  82000.,  83000.,  84000.,  85000.,  86000.,
   87000., 88000., 89000.,
    90000.,  91000.,  92000.,  93000.,  94000.,  95000.,  96000.,
   97000., 98000., 99000.};

/* hot atmosphere ambient temp array (deg R) */

float hot_temp[N_ATM_ALT] = {
   562.7f,  558.9f,  555.1f,  551.2f,  547.3f,  543.4f,  539.5f,
   535.5f,  531.5f,  527.5f,
   523.6f,  519.9f,  516.1f,  512.3f,  508.5f,  504.6f,  500.7f,
   496.8f,  492.8f,  488.9f,
   485.2f,  481.5f,  477.7f,  474.0f,  470.2f,  466.4f,  462.6f,
   458.7f,  454.8f,  451.0f,
   447.4f,  443.8f,  440.2f,  436.5f,  432.9f,  429.6f,  426.3f,
   423.0f,  419.6f,  416.2f,
   414.9f,  415.4f,  415.8f,  416.2f,  416.6f,  417.1f,  417.6f,
   418.0f,  418.5f,  419.0f,
   419.5f,  419.8f,  420.0f,  420.2f,  420.4f,  420.6f,  420.7f,
   420.9f,  421.1f,  421.3f,
   421.5f,  421.7f,  421.9f,  422.1f,  422.3f,  422.5f,  422.6f,
   423.0f,  423.6f,  424.2f,
   425.0f,  425.7f,  426.4f,  427.1f,  427.8f,  428.5f,  429.2f,
   429.9f,  430.6f,  431.3f,
   432.0f,  432.8f,  433.6f,  434.3f,  435.1f,  435.9f,  436.7f,
   437.5f,  438.4f,  439.2f,
   439.9f,  440.6f,  441.4f,  442.1f,  442.9f,  443.8f,  444.6f,
   445.4f,  446.3f,  447.2f};
```

```
/*  cold atmosphere ambient temp array (deg R) */

 float cold_temp[N_ATM_ALT] ={
     399.7f,  413.2f,  426.7f,  440.4f,  444.7f,  444.7f,  444.7f,
    444.7f, 444.7f, 444.7f,
     444.7f,  443.9f,  440.6f,  437.3f,  434.0f,  430.6f,  427.3f,
    423.9f, 420.5f, 447.0f,
     413.6f,  410.1f,  406.5f,  403.0f,  399.4f,  395.8f,  392.2f,
    388.6f, 384.9f, 381.1f,
     377.4f,  374.7f,  374.7f,  374.7f,  374.7f,  374.7f,  374.7f,
    374.7f, 374.7f, 374.7f,
     374.7f,  374.7f,  374.7f,  371.5f,  366.4f,  361.1f,  355.8f,
    350.4f, 345.0f, 340.5f,
     336.8f,  334.7f,  334.7f,  334.7f,  334.7f,  334.7f,  334.7f,
    334.7f, 334.7f, 334.7f,
     334.7f,  334.7f,  337.6f,  340.7f,  343.7f,  346.5f,  349.8f,
    351.9f, 354.4f, 356.9f,
     359.2f,  361.4f,  363.6f,  365.6f,  365.4f,  365.2f,  364.9f,
    364.7f, 364.4f, 364.1f,
     363.8f,  363.5f,  363.1f,  362.7f,  362.3f,  361.9f,  361.5f,
    361.2f, 360.8f, 360.4f,
     360.0f,  359.6f,  359.2f,  358.8f,  358.4f,  358.0f,  357.6f,
    357.1f, 356.7f, 356.3f};

/*  tropial atmosphere ambient temp array (deg R) */

   float trop_temp[N_ATM_ALT] = {
      549.5f, 545.6f,  541.7f,  537.8f,  534.0f,  530.1f,  526.2f,
       522.3f,  518.4f,  514.6f,
      510.7f, 506.8f,  502.9f,  499.1f,  495.2f,  491.3f,  487.5f,
       483.6f,  479.7f,  475.8f,
      472.0f, 468.1f,  464.2f,  460.4f,  456.5f,  452.7f,  448.8f,
       444.9f,  441.7f,  437.2f,
      433.4f, 429.5f,  425.6f,  421.8f,  417.9f,  414.1f,  410.2f,
       406.4f,  402.6f,  398.8f,
      395.1f, 391.4f,  387.7f,  384.1f,  380.5f,  376.9f,  373.4f,
       369.9f,  366.5f,  363.0f,
      359.6f, 356.3f,  352.9f,  349.6f,  348.6f,  350.7f,  352.9f,
       355.1f,  357.2f,  359.5f,
      361.7f, 363.9f,  366.1f,  368.4f,  370.7f,  373.0f,  375.3f,
       377.6f,  379.9f,  382.2f,
      384.2f, 385.5f,  386.8f,  388.1f,  389.4f,  390.7f,  392.1f,
       393.4f,  394.7f,  396.1f,
      397.4f, 398.8f,  400.1f,  401.5f,  402.8f,  404.2f,  405.5f,
       406.9f,  408.2f,  409.6f,
      410.9f, 412.2f,  413.6f,  404.9f,  416.3f,  417.6f,  418.9f,
       420.3f,  421.6f,  422.9f};

/* polar atmosphere ambient temp array (deg R) */

  float polar_temp[N_ATM_ALT] = {
       444.0f,   447.0f,   450.1f,   453.1f,   453.5f,   453.0f,
      452.4f,  451.9f,  451.3f,  450.8f,
       450.0f,   447.2f,   444.3f,   441.5f,   438.7f,   435.9f,
      433.0f,  430.2f,  427.4f,  424.5f,
       421.7f,   418.8f,   416.0f,   413.1f,   410.3f,   407.4f,
      404.5f,  401.7f,  398.8f,  395.9f,
       393.0f,   392.5f,   392.2f,   392.0f,   391.7f,   391.4f,
      391.2f,  390.9f,  390.7f,  390.4f,
```

```
        390.1f,    389.9f,    389.6f,    389.4f,    389.1f,    388.8f,
       388.6f,  388.3f,  388.1f,  387.8f,
        387.5f,    387.3f,    387.0f,    386.8f,    386.5f,    386.2f,
       386.0f,  385.7f,  385.5f,  385.2f,
        385.0f,    384.7f,    384.4f,    384.2f,    383.9f,    383.7f,
       383.4f,  383.2f,  382.9f,  382.6f,
        382.4f,    382.1f,    381.9f,    381.6f,    381.4f,    381.1f,
       380.9f,  380.6f,  380.3f,  380.1f,
        379.8f,    379.6f,    379.3f,    379.1f,    378.8f,    378.6f,
       378.3f,  378.3f,  378.3f,  378.3f,
        378.3f,    378.3f,    378.3f,    378.3f,    378.3f,    378.3f,
       378.3f,  378.3f,  378.3f,  378.3f};
//---------------------------------------------------------------
C_ADC_Model::C_ADC_Model()
{
      testMode = testmode_NONE;
      testTimer = 0;
}
//---------------------------------------------------------------
C_ADC_Model::~C_ADC_Model(){
}
//---------------------------------------------------------------
void C_ADC_Model::Init()
{

      test_flag = TRUE;

      /* Power up default values */
      E_ADC_C01.HBC_V = C_NOTVALID;
      E_ADC_C01.PBS_VALID = C_NOTVALID;
      E_ADC_C01.PRE_ST = (float)ADC_BARO_SET_NOM;
      dLastValidValueBaroPresSet = ADC_BARO_SET_NOM;
      E_ADC_C01.ALT_HBC = (float)E_FL_OUTSTATE.ALT;
      E_ADC_CONTROL.ADC_DataVal = C_VALID;

      InitBlock.Altitude           = E_FL_OUTSTATE.ALT;
      InitBlock.DtStep= FrameTime = C_ADC_MODEL_FRAME_TIME;
      InitBlock.PressureAltitude_TimeConst=0;
      ADC_INIT (InitBlock,&Output) ;
}
//---------------------------------------------------------------
void C_ADC_Model::Run()
{

      if (E_ADC_CONTROL.Isact == C_ACTIVE)
      {
            SetAdcInput (&Input);
            TestInput = Input;
            if ((E_ADC_C02.STEST_1_CMD == C_ACTIVE
                  || E_ADC_C02.STEST_2_CMD == C_ACTIVE
                  || E_ADC_C02.Trans_Test == C_ACTIVE)
                  && E_FL_OUTSTATE.Vt <= (70.0 * KN_2_FPS) )
                  /* Self test */
                  SetAdcSelfTest(&TestInput,&Output) ;
            else
            {   /* Normal mode */
                  ADC_UPDATE (Input, &Output) ;
                  UpdateAdcMainOutMember (&Output) ;
                  DisplayResults( "Normal Mode" );
```

```
            }
        }
}
//---------------------------------------------------------------------
void C_ADC_Model::ReadInputs(){

    int inputIndex = -1;
    try
    {
       inputIndex = 0;
       E_FL_OUTSTATE.ALT=StrToFloat(Avionics->altInput->Text );
       inputIndex++;
       E_FL_OUTSTATE.Vt = StrToFloat( Avionics->tasInput->Text );
       inputIndex++;
       E_ADC_C02.Baro_Pr_Set=StrToFloat(Avionics->presInput->Text
           );
       inputIndex++;
       InitBlock.TempDevDegC=StrToFloat(Avionics->tempDevInput-
            >Text);
       inputIndex++;
       InitBlock.PressureAltitude_TimeConst=StrToFloat(Avionics-
            >presAltTimeCInput->Text );
    }
    __except( 1 )
   {
        switch( inputIndex ){
           case 0: Avionics->altInput->Text = "10000"; break;
           case 1: Avionics->tasInput->Text = "500"; break;
           case 2: Avionics->presInput->Text = "29.92"; break;
           case 3: Avionics->tempDevInput->Text = "0"; break;
           case 4: Avionics->presAltTimeCInput->Text = "0"; break;
        }
   }

   InitBlock.Atmos_Type = Avionics->atmTypeInput->ItemIndex;

   switch( testMode ){
        case testmode_NONE: break;
        case testmode_STEST1:
                E_ADC_C02.STEST_1_CMD = C_ACTIVE; break;
        case testmode_STEST2:
                E_ADC_C02.STEST_2_CMD = C_ACTIVE; break;
        case testmode_TRANS:
                E_ADC_C02.Trans_Test = C_ACTIVE; break;
   }

}
//---------------------------------------------------------------------
void C_ADC_Model::UpdateAdcMainOutMember (Adc_Output *Output)
{
   static float Mach_old = 0.;

       if (test_flag == TRUE &&
              E_ADC_C02.STEST_1_CMD != C_ACTIVE &&
              E_ADC_C02.STEST_2_CMD != C_ACTIVE &&
              E_ADC_C02.Trans_Test != C_ACTIVE)
            // Self test is not active
            {
             test_flag = FALSE;
```

```
      E_ADC_C01.Stest_Reach = C_FALSE;
      E_ADC_C01.Self_test_1 = C_INACTIVE;
      E_ADC_C01.Self_test_2 = C_INACTIVE;
     }

/* Check for Injected error - Data is invalid (C01 & C06)*/
if(E_ADC_CONTROL.ADC_DataVal == C_NOTVALID)
 {
      /* Injected error Data is invalid*/
     E_ADC_C01.HP_PS_V = E_ADC_C06.HP_PS_V = C_NOTVALID;
     E_ADC_C01.HBC_V = E_ADC_C06.HBC_V = C_NOTVALID;
     E_ADC_C01.VT_ADR_V = E_ADC_C06.VT_ADR_V = C_NOTVALID;
     E_ADC_C01.MACH__V = E_ADC_C06.MACH__V = C_NOTVALID;
     E_ADC_C01.VC_QC_V = E_ADC_C06.VC_QC_V = C_NOTVALID;
     E_ADC_C01.PS_PS0_V = E_ADC_C06.PS_PS0_V = C_NOTVALID;
     E_ADC_C01.AOAT_V = E_ADC_C06.AOAT_V = C_NOTVALID;
     E_ADC_C01.TEM_ST_V = E_ADC_C06.TEM_ST_V = C_NOTVALID;
     E_ADC_C01.HPR_V = C_NOTVALID;
     E_ADC_C01.M_RATE_V = C_NOTVALID;
     E_ADC_C01.PRE_TOT_V = C_NOTVALID;
     E_ADC_C01.PBS_VALID = C_NOTVALID;
     E_ADC_C01.AOA_IND_V = C_NOTVALID;
     E_ADC_C01.TEM_TOT_V = C_NOTVALID;
     return;
 }

/* Set data to valid*/
E_ADC_C01.HP_PS_V = E_ADC_C06.HP_PS_V = C_VALID;
E_ADC_C01.VT_ADR_V = E_ADC_C06.VT_ADR_V = C_VALID;
E_ADC_C01.MACH__V = E_ADC_C06.MACH__V = C_VALID;
E_ADC_C01.VC_QC_V = E_ADC_C06.VC_QC_V = C_VALID;
E_ADC_C01.PS_PS0_V = E_ADC_C06.PS_PS0_V = C_VALID;
E_ADC_C01.AOAT_V = E_ADC_C06.AOAT_V = C_VALID;
E_ADC_C01.TEM_ST_V = E_ADC_C06.TEM_ST_V = C_VALID;
E_ADC_C01.HPR_V = C_VALID;
E_ADC_C01.M_RATE_V = C_VALID;
E_ADC_C01.PRE_TOT_V = C_VALID;
E_ADC_C01.PBS_VALID = C_VALID;
E_ADC_C01.AOA_IND_V = C_VALID;
E_ADC_C01.TEM_TOT_V = C_VALID;


/* Copy ADC data to output data base structures
E_ADC_C01.ALT_HP = E_ADC_C06.ALT_HP =
     (float)Output ->PressureAltitude;
E_ADC_C01.ALT_HBC = E_ADC_C06.ALT_HBC =
     (float)Output ->BaroCorrectedAltitude;
E_ADC_C01.TAS = E_ADC_C06.TAS =
     (float)(Output ->TAS * FPS_2_KN);

Mach_old = E_ADC_C01.MACH_NUM;
E_ADC_C01.MACH_NUM=E_ADC_C06.Mach_Num=(float)Output->Mach;


E_ADC_C01.AOA_TRUE = E_ADC_C06.AOA_TRUE =
     (float)(E_FL_OUTSTATE.alpha * RAD_2_DEG);

E_ADC_C01.Ps_Ps0 = E_ADC_C06.Ps_Ps0 =
   (float)Output ->PressureRatio;
```

```
E_ADC_C01.RHO_RHO0 = E_ADC_C06.AirDenR =
   (float)Output ->DensityRatio;
E_ADC_C01.TEM_ST=(float)(DegR_2_DegC(Output->Temp_static));
E_ADC_C06.StatTmp =  (float)Output ->Temp_static / 1.8f;
                                                  /* deg K */

E_ADC_C01.HP_RATE = (float)E_Fl_ADDIT.GrSpeedUp;

/* Mach Rate */
E_ADC_C01.MACH_RATE = (float)((Output ->Mach - Mach_old) /
   FrameTime * 60.); /* Mach/min */

E_ADC_C01.PRE_ST  =  (float)(Output  ->  Pressure_static  *
   PSF_2_InHG);
E_ADC_C01.PRE_TOT  =  (float)(Output  ->  TotalPressure*
   PSF_2_InHG);
E_ADC_C01.Qc_C01  =  (float)(Output  ->  ImpactPressure  *
   PSF_2_InHG);

/* Baro ..............*/
E_ADC_C01.ADC_PBS = E_ADC_C02.Baro_Pr_Set;

E_ADC_C01.AOA_IND  =  (float)(1.6893*E_ADC_C01.AOA_TRUE  -
   0.668); // From tests results

E_ADC_C01.PRE_S_IND = E_ADC_C01.PRE_ST;
E_ADC_C01.TEMP_TOT  =  (float)(DegR_2_DegC  (  Output  ->
   TotalTemp ));
E_ADC_C01.MACH_IND = (float)Output ->Mach;
E_ADC_C01.Qc_ind = E_ADC_C01.Qc_C01;

E_ADC_C06.Elap_time  =  (float)(E_SIM_CONTROL.FramesCtr  *
   FrameTime / 3600.); /* Hour */

/* Check Outputs is valid */
if (E_ADC_C01.ALT_HP < -1500. || E_ADC_C01.ALT_HP > 80000.)
     E_ADC_C01.HP_PS_V = E_ADC_C06.HP_PS_V = C_NOTVALID ;

if (E_ADC_C01.ALT_HBC<-1500.||E_ADC_C01.ALT_HBC > 80000.)
     E_ADC_C01.HBC_V = E_ADC_C06.HBC_V = C_NOTVALID;

if (E_ADC_C01.TAS < 70. || E_ADC_C01.TAS > 1740.4 ||
   E_ADC_C01.RHO_RHO0<0.035529||E_ADC_C01.RHO_RHO0> 1.0446)
     E_ADC_C01.VT_ADR_V = C_NOTVALID;
if (E_ADC_C06.TAS < 70. || E_ADC_C06.TAS > 1720. ||
     E_ADC_C06.AirDenR < 0.035529 ||
     E_ADC_C06.AirDenR > 1.0446)
   E_ADC_C06.VT_ADR_V = C_NOTVALID;

if (E_ADC_C01.MACH_NUM < 0.1 || E_ADC_C01.MACH_NUM > 3.)
   E_ADC_C01.MACH__V = E_ADC_C06.MACH__V = C_NOTVALID;

if (E_ADC_C01.CAS < 50. || E_ADC_C01.CAS > 1000.)
  E_ADC_C01.VC_QC_V = C_NOTVALID;
if (E_ADC_C06.CAS < 50. || E_ADC_C06.CAS > 975.)
  E_ADC_C06.VC_QC_V = C_NOTVALID;

if (E_ADC_C01.Ps_Ps0 < 0.0272536 ||
    E_ADC_C01.Ps_Ps0 > 1.05541)
```

```
        E_ADC_C01.PS_PS0_V = E_ADC_C06.PS_PS0_V = C_NOTVALID;

      if (E_ADC_C01.AOA_TRUE < -15 || E_ADC_C01.AOA_TRUE > 30)
        E_ADC_C01.AOAT_V = E_ADC_C06.AOAT_V = C_NOTVALID ;

      if (E_ADC_C01.TEM_ST < -100. || E_ADC_C01.TEM_ST > 50.)
        E_ADC_C01.TEM_ST_V = E_ADC_C06.TEM_ST_V = C_NOTVALID;

       /* C01 */
      if (E_ADC_C01.HP_RATE < -1600. ||
          E_ADC_C01.HP_RATE > 1600.)
        E_ADC_C01.HPR_V = C_NOTVALID;

      if (E_ADC_C01.MACH_RATE < -2.5 ||
          E_ADC_C01.MACH_RATE > 2.5)
        E_ADC_C01.M_RATE_V = C_NOTVALID;

      if (E_ADC_C01.PRE_TOT < 0.7 ||
          E_ADC_C01.PRE_TOT > 102.)
        E_ADC_C01.PRE_TOT_V = C_NOTVALID;

      if (E_ADC_C01.AOA_IND > 40.)
        E_ADC_C01.AOA_IND = 40.;

     else if(E_ADC_C01.AOA_IND < -10)
        E_ADC_C01.AOA_IND = -10.;

      if (E_ADC_C01.AOA_IND < -10 || E_ADC_C01.AOA_IND > 40)
        E_ADC_C01.AOA_IND_V = C_NOTVALID;

      if (E_ADC_C01.TEMP_TOT < -100. || E_ADC_C01.TEMP_TOT>350.)
        E_ADC_C01.TEM_TOT_V = C_NOTVALID;

}
//---------------------------------------------------------------------
void C_ADC_Model::SetAdcInput (Adc_Input *Input)
{

      /* Fill Input data stucture */
      Input -> Altitude    = E_FL_OUTSTATE.ALT ; // ft
      Input -> TAS         = E_FL_OUTSTATE.Vt ;  // ft/sec


      if ((E_ADC_C02.Baro_Pr_Set <= ADC_BARO_SET_MAX)
       &&(E_ADC_C02.Baro_Pr_Set >= ADC_BARO_SET_MIN))
      {     // Baro correction valid
            Input->BaroAltitudeCorrection = E_ADC_C02.Baro_Pr_Set;
            // InHG
            E_ADC_C01.HBC_V = E_ADC_C06.HBC_V = C_VALID;
            E_ADC_C01.PBS_VALID = C_VALID;
            dLastValidValueBaroPresSet = E_ADC_C02.Baro_Pr_Set;
    }
  // Baro correction Not valid, use last valid value
      else
      {
         Input->BaroAltitudeCorrection=
                  dLastValidValueBaroPresSet;
         E_ADC_C01.HBC_V = E_ADC_C06.HBC_V = C_NOTVALID;
         E_ADC_C01.PBS_VALID = C_NOTVALID;
```

```
    }

}
//-------------------------------------------------------------------
void    C_ADC_Model::SetAdcSelfTest(Adc_Input    *Input,Adc_Output
*Output)
{

    /* Self Test shall be inhibited if TAS > 70 Knots */
    if ( E_FL_OUTSTATE.Vt > 70 *KN_2_FPS)
     return;

    test_flag = TRUE;
    E_ADC_C01.TestFailed = C_FALSE;
    E_ADC_C01.HP_PS_V = E_ADC_C06.HP_PS_V = C_NOTVALID ;
    E_ADC_C01.HBC_V = E_ADC_C06.HBC_V = C_NOTVALID;
    E_ADC_C01.VT_ADR_V = E_ADC_C06.VT_ADR_V = C_NOTVALID;
    E_ADC_C01.MACH__V = E_ADC_C06.MACH__V = C_NOTVALID;
    E_ADC_C01.VC_QC_V = E_ADC_C06.VC_QC_V = C_NOTVALID;
    E_ADC_C01.PS_PS0_V = E_ADC_C06.PS_PS0_V = C_NOTVALID;
    E_ADC_C01.AOAT_V = E_ADC_C06.AOAT_V = C_NOTVALID;
    E_ADC_C01.TEM_ST_V = E_ADC_C06.TEM_ST_V = C_NOTVALID;
    E_ADC_C01.HPR_V = C_NOTVALID;
    E_ADC_C01.M_RATE_V = C_NOTVALID;
    E_ADC_C01.PRE_TOT_V = C_NOTVALID;
    E_ADC_C01.PBS_VALID = C_NOTVALID;
    E_ADC_C01.AOA_IND_V = C_NOTVALID;
    E_ADC_C01.TEM_TOT_V = C_NOTVALID;

    if (E_ADC_C02.STEST_1_CMD == C_ACTIVE)
    {     /* Self test 1 */
       E_ADC_C01.Self_test_1 = C_ACTIVE;

       Input->Altitude    = 10000 ; // ft
       Input->TAS         = 500. ;      // ft/sec
       Input->BaroAltitudeCorrection = ADC_BARO_SET_NOM;
       ADC_UPDATE (*Input, Output) ;
       UpdateAdcMainOutMember (Output) ;
       E_ADC_C01.AOA_TRUE = 0.0; /* deg */
       E_ADC_C01.AOA_IND = 0.0;  /* deg */
       E_ADC_C01.Stest_Reach = C_TRUE;

          DisplayResults( "S-Test-1 Results:" );
          E_ADC_C02.STEST_1_CMD = C_INACTIVE;
    }


    if (E_ADC_C02.STEST_2_CMD == C_ACTIVE)
    {     /* Self test 2 */

       E_ADC_C01.Self_test_2 = C_ACTIVE;
        Input -> Altitude    = 20000 ;         // ft
        Input -> TAS         = 900.000 ;       // ft/sec
        Input -> BaroAltitudeCorrection = ADC_BARO_SET_NOM;
        ADC_UPDATE (*Input, Output) ;
        UpdateAdcMainOutMember (Output) ;
        E_ADC_C01.AOA_TRUE = 6.0; /* deg */
        E_ADC_C01.AOA_IND = 6.0;  /* deg */
        E_ADC_C01.Stest_Reach = C_TRUE;
```

```
        DisplayResults( "S-Test-2 Results:" );
        E_ADC_C02.STEST_2_CMD = C_INACTIVE;
    }

    if (E_ADC_C02.Trans_Test == C_ACTIVE)
    {
        DisplayResults( "Trans-Test Results:" );
        E_ADC_C02.Trans_Test = C_INACTIVE;
    }

    if( testTimer == 0 )
    {
        testMode = testmode_NONE;
    }
}
//---------------------------------------------------------------
int   C_ADC_Model::ADC_INIT   (Init_Adc_input   InitAtm,Adc_Output
*modelOutput)
{

    double  Temp_Dev_DegR;
                    /* temperature deviation from std (deg R) */
    int ret_val;    /* function return value */

/* Convert temperature dev from deg celsius to deg rankine */
    Temp_Dev_DegR = (InitAtm.Temp_Dev_DegC) * 1.8;

/* calculate pres,temp,airdensity & speed of sound at sea level */
/* default to standard day */
gr_ADC_model_AtmosData.Atmos_Type = InitAtm.Atmos_Type;
ret_val = 0;

switch (gr_ADC_model_AtmosData.Atmos_Type)
{
    case HOT_ATMOS:                 /* hot day */
    {
     gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 562.7;
     gr_ADC_model_AtmosData.SeaLevelRho = 2.19e-03;
     gr_ADC_model_AtmosData.SeaLevelPressure = 2116.22;
     gr_ADC_model_AtmosData.SeaLevelSound = 1162.84;
     break;
    }

    case TROPICAL_ATMOS:            /* tropical day */
    {
     gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 562.7;
     gr_ADC_model_AtmosData.SeaLevelRho = 2.246e-03;
     gr_ADC_model_AtmosData.SeaLevelPressure = 2116.22;
     gr_ADC_model_AtmosData.SeaLevelSound = 1162.84;
     break;
        }

    case POLAR_ATMOS:               /* polar day */
    {
    gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 444.0;
    gr_ADC_model_AtmosData.SeaLevelRho = 2.817e-03;
    gr_ADC_model_AtmosData.SeaLevelPressure   = 2140.7;
    gr_ADC_model_AtmosData.SeaLevelSound = 1032.94;
    break;
```

```
   }

   case COLD_ATMOS:                    /* cold day */
   {
      gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 399.7;
      gr_ADC_model_AtmosData.SeaLevelRho = 3.09e-03;
      gr_ADC_model_AtmosData.SeaLevelPressure = 2116.22;
      gr_ADC_model_AtmosData.SeaLevelSound = 980.05;
      break;
      }

   case STD_ATMOS:                     /* std day */
   {
      gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 518.69;
      gr_ADC_model_AtmosData.SeaLevelRho = 0.0023769;
      gr_ADC_model_AtmosData.SeaLevelPressure = 2116.22;
      gr_ADC_model_AtmosData.SeaLevelSound = 1116.44;
      break;
   }
   default:
   {
      ret_val = WRONG_ATMOS_TYPE_USE_STD;
      gr_ADC_model_AtmosData.Atmos_Type = STD_ATMOS;
      gr_ADC_model_AtmosData.SeaLevelTemp_no_dev = 518.69;
      gr_ADC_model_AtmosData.SeaLevelRho = 0.0023769;
      gr_ADC_model_AtmosData.SeaLevelPressure = 2116.22;
      gr_ADC_model_AtmosData.SeaLevelSound = 1116.44;
      break;
   }
}

/* Sea level temperature                                          */
gr_ADC_model_AtmosData.SeaLevelTemp =
      gr_ADC_model_AtmosData.SeaLevelTemp_no_dev + Temp_Dev_DegR;

/* normalize temperature deviation to sea level temp for the  */
/* temperature ratio deviation.                                   */

gr_ADC_model_AtmosData.DeltaTempDevRatio    =    Temp_Dev_DegR    /
      gr_ADC_model_AtmosData.SeaLevelTemp_no_dev;

 /* init Altitude filter */
modelOutput->PressureAltitude = InitAtm.Altitude;
gr_ADC_model_AtmosData.DtStep = InitAtm.DtStep;
gr_ADC_model_AtmosData.PressureAltitude_TimeConst
                    = InitAtm.PressureAltitude_TimeConst;

return(ret_val);
}
//-----------------------------------------------------------------

int     C_ADC_Model::ADC_UPDATE(Adc_Input     modelInput,Adc_Output
*modelOutput)
{
 static double qpasl1, oper;
 static double t_rsigma;
 static double mach_sq, f1, sqrt_rsigma /*,adr_basis1*/ ;
 static double  sound;         /* Speed of sound [ft/sec] */
```

```
double ttheta;          /*Temperature ratio(non-dimensional)*/
double pdelta;          /* Pressure ratio (non-dimensional)*/
double rsigma;          /* Density ratio (non-dimensional) */
static int atm_alt_pt = 0;  /* Atmos alt breakpoint tracker */
double baro_set;        /* Baro Pressure [psf] */
double th_baro;         /* Temperature for Baro Pressure[deg R] */
double delta_alt;       /* Delta altitude for Barocorrection */

/* Calculate pres ratio & temp ratio for standard day */

      if (modelInput.Altitude <= 36152.0)
      {
        ttheta = 1.0 - 6.87528e-06 * modelInput.Altitude;
        pdelta = pow(ttheta, 5.256152);
      }
      else if (modelInput.Altitude <= 65824.0)
      {
        ttheta = 0.751865;
        pdelta = 0.223361 / exp ( ( modelInput.Altitude - 36089.2)
          * 4.80637e-05 );
      }
      else
      {
         ttheta = 0.10578e-05 * ( modelInput.Altitude - 65616.8 )
          + 0.751865;
         pdelta = 0.751865 / ttheta;
         pdelta = 0.0540328 * pow(pdelta, 34.203357);
      }

/*  Calculate temperature ratio.
/*   Non-std atmospheric models based on MIL-STD-210A; pressure
ratio same  as standard day except for the polar day model. */

      switch (gr_ADC_model_AtmosData.Atmos_Type)
      {
            case HOT_ATMOS:
            {
            /* hot day */
                  f1    =    (double)F1V((float)modelInput.Altitude,
atmos_alt, hot_temp, N_ATM_ALT, &atm_alt_pt, 1);
                  ttheta = f1  / (double)hot_temp[0];
                  break;
            }

            case TROPICAL_ATMOS:
            {
            /* tropical day */
                  f1    =    (double)F1V((float)modelInput.Altitude,
atmos_alt, trop_temp, N_ATM_ALT, &atm_alt_pt, 1);
                  ttheta = f1 / (double)trop_temp[0];
                  break;
            }

            case POLAR_ATMOS:
            {
            /* polar day */

                  pdelta     =     pdelta     *     2116.22     /
gr_ADC_model_AtmosData.SeaLevelPressure;
```

```
                f1    =    (double)F1V((float)modelInput.Altitude,
atmos_alt, polar_temp, N_ATM_ALT, &atm_alt_pt, 1);
                ttheta = f1  / (double)polar_temp[0];
                break;
          }

          case COLD_ATMOS:
          {
          /* cold day */
                f1    =    (double)F1V((float)modelInput.Altitude,
atmos_alt, cold_temp, N_ATM_ALT, &atm_alt_pt, 1);
                ttheta = f1  / (double)cold_temp[0];
                break;
          }

          default:
          {
                break;
          }
     }

/* temp ratio corrected for temperature deviation */

     ttheta = ttheta + gr_ADC_model_AtmosData.DeltaTempDevRatio;

/* density ratio  */
     rsigma = pdelta / ttheta;
     if (rsigma < 1.0e-20)
          sqrt_rsigma = 1.0e-20;
     else
          sqrt_rsigma = sqrt(rsigma);
     t_rsigma = 0.5925*sqrt_rsigma;

 /* Atmospheric characteristics  */

     modelOutput->Temp_static=
        gr_ADC_model_AtmosData.SeaLevelTemp_no_dev * ttheta;
     modelOutput->Rho=        gr_ADC_model_AtmosData.SeaLevelRho*
        rsigma;
     modelOutput->Pressure_static=
        gr_ADC_model_AtmosData.SeaLevelPressure * pdelta;
     sound    = 49.021*sqrt(modelOutput->Temp_static);
     modelOutput->TAS = modelInput.TAS;
     modelOutput->Mach    = modelInput.TAS/sound;
     mach_sq = modelOutput->Mach * modelOutput->Mach;
     modelOutput->DensityRatio = rsigma;
     modelOutput->PressureRatio = pdelta;
     modelOutput->DynamicPressure  =  0.5  *  modelOutput->Rho  *
        modelInput.TAS*modelInput.TAS;
     modelOutput->TotalTemp=modelOutput->Temp_static*(1.0+  0.20*
        mach_sq);

/* Calculate equivalent,calibrated airspeed and impact pressure */
     modelOutput->Veas = modelInput.TAS * sqrt_rsigma;

     if (modelOutput->Mach != 0.0)
     {
          if (modelOutput->Mach <= 1.0)
```

```
            modelOutput->ImpactPressure=(pow((1.0+
0.2*mach_sq),3.5) - 1.0) * modelOutput->Pressure_static;
          else
            modelOutput->ImpactPressure                              =
((166.9*mach_sq)/(pow(7.0 - 1.0/(mach_sq),2.5)) - 1.)*

modelOutput->Pressure_static;
      }
      else
            modelOutput->ImpactPressure = 0.0;

      qpasl1=modelOutput->ImpactPressure/
gr_ADC_model_AtmosData.SeaLevelPressure + 1.0;

      modelOutput->Vcas = /*1479.12 for Knots*/

2496.4357328*sqrt(pow(qpasl1,0.285714) - 1.0) ;

      modelOutput->TotalPressure = modelOutput->Pressure_static +
modelOutput->ImpactPressure;

      if (modelOutput->ImpactPressure > 1889.64)  /* Mach > 1.0 */
      {
        oper = qpasl1*pow(7. -
                        pow(gr_ADC_model_AtmosData.SeaLevelSound
/ modelOutput->Vcas,2.0),2.5);

        if (oper < 0.0)
              oper = 0.1;
        modelOutput->Vcas = 51.1987*sqrt(oper) * KN_2_FPS;
                            /* 51.1987 is define for Knots */
      }

 /* Altitude filter */
  if( gr_ADC_model_AtmosData.PressureAltitude_TimeConst > 0.0)
      modelOutput->PressureAltitude=modelOutput-
>PressureAltitude + (gr_ADC_model_AtmosData.DtStep
*gr_ADC_model_AtmosData.PressureAltitude_TimeConst)*
(modelInput.Altitude - modelOutput->PressureAltitude) ;

  else
      modelOutput->PressureAltitude = modelInput.Altitude;

      modelOutput->DensityAltitude= modelOutput->PressureAltitude;


  /* Checking the baro set validity */
  if (modelInput.BaroAltitudeCorrection > 0.0)
  {
            /* ADC receives valid signal & compute the barocor*/
        baro_set = modelInput.BaroAltitudeCorrection*InHG_2_PSF;

        th_baro  = gr_ADC_model_AtmosData.SeaLevelTemp /
 (pow(baro_set/gr_ADC_model_AtmosData.SeaLevelPressure,
                                    1./FL_PR_POW));
        delta_alt=-(th_baro-gr_ADC_model_AtmosData.SeaLevelTemp)
/FL_R0;
 }
```

```
 else
        delta_alt = 0.0;

 if (delta_alt == 0.0)
            /* Baro correction is not needed */
      modelOutput->BaroCorrectedAltitude = modelInput.Altitude ;
 else
            /* Baro correction is needed */
      modelOutput->BaroCorrectedAltitude=modelInput.Altitude+
delta_alt;

 return 1;
}
//---------------------------------------------------------------------
void C_ADC_Model::CheckActivity(){

        E_ADC_CONTROL.Isact =
                ( Avionics->powerInput->Checked == 1 ) &&
                ( Avionics->commFailInput->Checked == 0 );
}
//---------------------------------------------------------------------
void C_ADC_Model::DisplayResults( AnsiString resStr ){

      AnsiString namesStr = "";
      AnsiString valuesStr = "";

      #define ADDLINE( x ) namesStr += #x"\n";
        valuesStr += FloatToStrF( Output.x, ffFixed, 6, 4 )+"\n"

);



      ADDLINE( Temp_static );
      ADDLINE( Pressure_static );
      ADDLINE( Rho );
      ADDLINE( TAS );
      ADDLINE( Mach );
      ADDLINE( DynamicPressure );
      ADDLINE( Veas );
      ADDLINE( ImpactPressure );
      ADDLINE( Vcas );
      ADDLINE( TotalTemp );
      ADDLINE( TotalPressure );
      ADDLINE( DensityRatio );
      ADDLINE( PressureRatio );
      ADDLINE( DensityAltitude );
      ADDLINE( PressureAltitude );
      ADDLINE( BaroCorrectedAltitude );

      Avionics->ListBox1->Items->SetText( namesStr.c_str() );
      Avionics->ListBox2->Items->SetText( valuesStr.c_str() );
}
```

EK-2 Taktik Hava Seyrüsefer Model Kaynak Kodu

```
#include <math.h>
#include "main.h"
#include "caraModel.h"

//---------------------------------------------------------------------
C_CARA_Model::C_CARA_Model(){
}

C_CARA_Model::~C_CARA_Model(){
}

void C_CARA_Model::Run()
{
      if (Avionics->startButton->Caption == "STOP")
      {
            updateModelInput();
            updateCARAMode();
            caraCycle();

      // update flight parameters is required
        if( Avionics->altUpdateInput->Checked )
        {
            if((flight_altitude>0.0)&& (flight_altitude<60000.0))
            {
               flight_altitude += flight_alt_rate * 0.01;
               Avionics->altScroll->Position=(int)
                                              flight_altitude;
               Avionics->altDisplay->Caption = IntToStr( Avionics-
                                          >altScroll->Position);
            }
        }

        updateModelOutput();
      }
      else
      {
        Avionics->altOutput->Caption = "XXXX";
        Avionics->altRateOutput->Caption = "XX";
        Avionics->caraModeOut->Caption = "OFF" ;
        Avionics->sAltInvalidOut->Color = clMaroon;
        Avionics->sAltInvalidOut->Font->Color = clRed;
        Avionics->sInIbitOut->Color = clGreen;
        Avionics->sInIbitOut->Font->Color = clLime;
        Avionics->sAltFailOut->Color = clMaroon;
        Avionics->sAltFailOut->Font->Color = clRed;
        Avionics->sAltLowOut->Color = clMaroon;
        Avionics->sAltLowOut->Font->Color = clRed;
      }
}

void C_CARA_Model::startWarmup()
{

    caraMode = caramode_WARMUP;
    fCARAInWarmup = true;
    warmupTimer = C_WARMUP_TIME;
```

```
    modelOutput.altitudeInvalidSignal = true;
    modelOutput.altitudeLowSignal = false;
}

void C_CARA_Model::endWarmup()
{

      fCARAInWarmup = false;
        warmupTimer = 0;

        switch( modelInput.caraSwitch )
        {
              case caraswitch_NORMAL:
                    caraMode = caramode_NORMAL; break;

              case caraswitch_STANDBY:
                    caraMode = caramode_STANDBY; break;
        }
}

void C_CARA_Model::updateCARAMode()
{

      if(   ( modelInput.caraPower != S_OFF ) &&
            ( modelInput.caraMalfunction != malf_CARA_COMM ) )
      {

             if( modelInput.caraSwitch != prevSwitchState )
             {
              switch( modelInput.caraSwitch )
              {
                  case caraswitch_OFF:
                        caraMode = caramode_OFF;
                  break;

                  case caraswitch_STANDBY:
                        if( prevSwitchState == caraswitch_OFF )
                        {
                              startWarmup();
                        }
                        else
                        caraMode = caramode_STANDBY;
                  break;

                  case caraswitch_NORMAL:
                        if( prevSwitchState ==caraswitch_STANDBY )
                        {
                              if( fCARAInWarmup == false )
                              caraMode = caramode_NORMAL;
                        }
                        else if(prevSwitchState== caraswitch_OFF )
                              startWarmup();
                               //caraMode = caramode_STANDBY;
                  break;
              }
            }
            prevSwitchState = modelInput.caraSwitch;
        }
        else
```

```
        {
             caraMode = caramode_OFF;
        }
}

void C_CARA_Model::caraCycle()
{

      if( fCARAInWarmup )
        { // CARA in warmup mode
            warmupCycle();
        }
        else
        { // CARA is not in warmup mode
            if( fIBITRequested ) // IBIT requested
                  ibitCycle();
            else
            { // IBIT is not requested
                  modelOutput.altitudeLowReferenceValue=
                        modelInput.altLowReferenceRequest;
                  if( modelInput.ibitRequest )
                   // IBIT request from GUI
                          startIBIT();
                  else
                  { // no IBIT request from GUI
                       if( caraMode == caramode_STANDBY )
                       {
                          modelOutput.altitudeInvalidSignal=true;
                          modelOutput.altitudeLowSignal = false;
                       }
                       else
                          caraCalculate();

                  }
            }
        }

}

void C_CARA_Model::Init(){
}

void C_CARA_Model::updateModelInput()
{

    // read cara mode switch position from GUI
    modelInput.caraSwitch=(CARASwitch_Type)  Avionics->caraSwitch-
                  >Position;

    // read master switch position from GUI
    if( Avionics->masterSwitch->Checked )
          modelInput.masterSwitch = S_ON;
    else
          modelInput.masterSwitch = S_OFF;

    // read cara power state from GUI
    if( Avionics->caraPower->Checked )
        modelInput.caraPower = S_ON;
    else
```

```
        modelInput.caraPower = S_OFF;

    if( Avionics->altUpdateInput->Checked == false )
          flight_altitude = (float) Avionics->altScroll->Position;

    flight_alt_rate = (float) Avionics->altRateScroll->Position;
    flight_pitch = (float) Avionics->pitchScroll->Position;
    flight_roll = (float) Avionics->rollScroll->Position;

}
//---------------------------------------------------------------------
void C_CARA_Model::warmupCycle()
{

      if( warmupTimer > 0 )
      {
              warmupTimer--;
                if( warmupTimer == 0 )
                  endWarmup();
      }
}
//---------------------------------------------------------------------
void C_CARA_Model::startIBIT(){

    modelOutput.radarAltitude = defaultRadarAltitude;
    modelOutput.altitudeRate = defaultAltitudeRate;
    modelOutput.ibitInProgressSignal = true;
    modelOutput.altitudeInvalidSignal = true;
    fIBITRequested = true;
    ibitTimer = C_IBIT_TIMER;
}
//---------------------------------------------------------------------
void C_CARA_Model::endIBIT()
{

    modelOutput.radarAltitude = (int) flight_altitude;
    modelOutput.altitudeRate = (int) flight_alt_rate;
    modelOutput.ibitInProgressSignal = false;
    modelInput.ibitRequest = false;
    fIBITRequested = false;
    modelOutput.altitudeInvalidSignal = false;
    if( modelInput.caraMalfunction == malf_CARA_ALT_FAIL )
      modelOutput.altimeterFailSignal = true;
    else
      modelOutput.altimeterFailSignal = false;

}
//---------------------------------------------------------------------
void C_CARA_Model::ibitCycle()
{

      if( ibitTimer > 0 )
      {
              ibitTimer--;

              if( ibitTimer == 0 )
              { // ibit was completed
                  endIBIT();
              }
```

```
      }

}
//-----------------------------------------------------------------
// calculates the radar altitude, altitude rate, altitude low
signal, and
// altitude valid signals from current state of the aircraft.
//-----------------------------------------------------------------
void C_CARA_Model::caraCalculate()
{

      if( modelInput.caraMalfunction != malf_CARA_STUCK )
        {

            bool fPitchIsLessThan45 = fabs(flight_pitch) < 45;
            bool fRollIsLessThan65 = fabs(flight_roll) < 65;
            bool fPitchIsMoreThan45 = fabs(flight_pitch) >= 45;
            bool fPitchIsLessThan50 = fabs(flight_pitch) < 50;
            bool fRollIsMoreThan65 = fabs(flight_roll) >= 65;
            bool fRollIsLessThan70 = fabs(flight_roll) < 70;
            bool fPitchIn45_50 = false;
            bool fRollIn65_70 = false;

            if( fPitchIsLessThan45 && fRollIsLessThan65 )
            {
                modelOutput.altitudeInvalidSignal = false;
                modelOutput.radarAltitude = (int) flight_altitude;
                modelOutput.altitudeRate = (int) flight_alt_rate;
                if((unsigned int)flight_altitude< modelInput.alt
                            LowReferenceRequest )
                   modelOutput.altitudeLowSignal = true;
                else
                   modelOutput.altitudeLowSignal = false;
            }
            else
            {

                fPitchIn45_50=fPitchIsMoreThan45&
                                        fPitchIsLessThan50;
                fRollIn65_70=fRollIsMoreThan65&
                                        fRollIsLessThan70;
                if( fPitchIn45_50 || fRollIn65_70 )
                {
                     modelOutput.altitudeInvalidSignal=(rand()&1)?
                                                  true:false;
                }
                else
                {
                        modelOutput.altitudeInvalidSignal = true;

                }
            }
}
//-----------------------------------------------------------------
-----------
void C_CARA_Model::updateModelOutput()
{

      if( modelOutput.altitudeInvalidSignal )
```

```
        {
              Avionics->sAltInvalidOut->Color = clRed;
                Avionics->sAltInvalidOut->Font->Color = clMaroon;
        }
        else
        {
              Avionics->sAltInvalidOut->Color = clMaroon;
                Avionics->sAltInvalidOut->Font->Color = clRed;
        }

        if( modelOutput.altitudeLowSignal )
        {
              Avionics->sAltLowOut->Color = clRed;
              Avionics->sAltLowOut->Font->Color = clMaroon;
        }
        else
        {
              Avionics->sAltLowOut->Color = clMaroon;
                Avionics->sAltLowOut->Font->Color = clRed;
        }

        if( modelOutput.altimeterFailSignal )
        {
              Avionics->sAltFailOut->Color = clRed;
              Avionics->sAltFailOut->Font->Color = clMaroon;

        }
        else
        {
              Avionics->sAltFailOut->Color = clMaroon;
              Avionics->sAltFailOut->Font->Color = clRed;

        }

        if( modelOutput.ibitInProgressSignal )
        {
              Avionics->sInIbitOut->Color = clLime;
              Avionics->sInIbitOut->Font->Color = clGreen;
        }
        else
        {
              Avionics->sInIbitOut->Color = clGreen;
              Avionics->sInIbitOut->Font->Color = clLime;

        }

        if(   modelOutput.altitudeInvalidSignal   ||   caraMode   ==
caramode_OFF )
        {
              Avionics->altOutput->Caption = "XXXX";
              Avionics->altRateOutput->Caption = "XX";
        }
        else
        {
              Avionics->altOutput->Caption=IntToStr(
modelOutput.radarAltitude );
              Avionics->altRateOutput->Caption=IntToStr(
modelOutput.altitudeRate );
        }
```

```
        switch( caraMode )
        {
              case caramode_OFF:
                Avionics->caraModeOut->Caption = "OFF"; break;
                case caramode_WARMUP:
                Avionics->caraModeOut->Caption = "WARMUP"; break;
                case caramode_STANDBY:
                Avionics->caraModeOut->Caption = "STANDBY"; break;
                case caramode_NORMAL:
                Avionics->caraModeOut->Caption = "NORMAL"; break;
        }
}
```

EK-3 Yükseklikölçer Modeli Kaynak Kodu

```
#include <math.h>
#include "main.h"
#include "tacanModel.h"

//---------------------------------------------------------------------
C_TACAN_Model::C_TACAN_Model(){
}

C_TACAN_Model::~C_TACAN_Model(){
}

void C_TACAN_Model::Run(){
 updateModelInput();
 processInputs();
   if( locals.mode == mode_OFF )
   {
      if( locals.warmupTimer > 0 )
      {
            locals.warmupTimer--;
            if( locals.warmupTimer == 0 )
            {
                  locals.mode = modelInput.modeSwitch;
            }
      }
   }
   else
   {
      if( locals.ibitTimer )
      {
                  ibitCycle();
      }
      else
      {
            tacanCalculate();
      }
   }

   updateModelOutput();
}

void C_TACAN_Model::generateRandomStations(){

      for( int i=0; i<C_MAX_TACAN_NUM; i++ )
      {
        stations[i].position.longitude=C_WEST_LON+(C_EAST_LON-
            C_WEST_LON) * rand() / RAND_MAX;
        stations[i].position.latitude=C_SOUTH_LAT + (C_NORTH_LAT-
            C_SOUTH_LAT) * rand() / RAND_MAX;
        stations[i].type=(rand() & 1) ? tacan_air : tacan_ground;
        stations[i].channel.no = (10 * (i+1)) & 127;
        stations[i].channel.band = (rand() & 1)? band_X : band_Y;
        stations[i].channel.callsign.letter1 = rand() % 26;
        stations[i].channel.callsign.letter2 = rand() % 26;
        stations[i].channel.callsign.letter3 = rand() % 26;
```

```
        stations[i].occlusion = not_occluded;
        str = char( 0x30 + i );
        str += ": ";
        str += char( 'A' + stations[i].channel.callsign.letter1 );
        str += char( 'A' + stations[i].channel.callsign.letter2 );
        str += char( 'A' + stations[i].channel.callsign.letter3 );
        str += " ";
        str += IntToStr( stations[i].channel.no );
        str += char( (stations[i].channel.band-1) + 'X' );
        str += " ";
        str += (stations[i].type == tacan_air) ? "air" : "ground";
        Avionics->stationSelectE->Items->Add( str );
    }

    Avionics->stationSelectE->ItemIndex = 0;
}

void C_TACAN_Model::Init(){
}

    TacanToFrom_Type C_TACAN_Model::calculateToFrom(){

    TacanToFrom_Type toFrom;
    ownship.orientation.heading=StrToInt(Avionics->HeadingAcEn-
            >Text);

    if( (270<fabs( ownship.orientation.heading -locals.bearing
            ))||     (fabs(     ownship.orientation.heading-
            locals.bearing )< 90.0 ))
          toFrom = tofrom_TO;
    else
          toFrom = tofrom_FROM;

    return toFrom;
}

void C_TACAN_Model::updateModelInput(){   //RunMode_Type runMode;

   if (Avionics->startButton->Caption == "STOP")
   {

    modelInput.modeSwitch = (TacanMode_Type) Avionics->modeSwE-
            >ItemIndex;
    modelInput.channelNo = StrToInt( Avionics->channelE->Text );
    modelInput.bandSwitch   =   (Band_Type)   (Avionics->bandE-
            >ItemIndex+1);
    modelInput.course = StrToInt( Avionics->courseE->Text );
    modelInput.ACLatitude   =   StrToFloat   (Avionics->LatAcEn-
            >Text);
    modelInput.ACLongitude   =   StrToFloat   (Avionics->LonAcEn-
            >Text);
    modelInput.ACAltitude   =   StrToFloat   (Avionics->AltAcEn-
            >Text);
    modelInput.ACHeading  =    StrToInt(  Avionics->HeadingAcEn-
            >Text);
    if (  Avionics->ObstructInput->Checked)
      lockedStation.occlusion = occluded;
    else
      lockedStation.occlusion = not_occluded;
```

```
    }
    else if (Avionics->startButton->Caption == "START")
    {
         lockedStation.type = tacan_none;
         Avionics->bearingD->Caption = "0";
         Avionics->toFromD->Caption = "0";
         Avionics->courseDevD->Caption = "0";
         AnsiString callsign = "";
         callsign += char('A' );
         callsign += char('A' );
         callsign += char('A' );
         Avionics->callsignD->Caption = callsign;

    }
}

void C_TACAN_Model::updateModelOutput(){

    if ((lockedStation.type==tacan_none)|| (lockedStation.occlusion
        == occluded))
    {

         lockedStation.type = tacan_none;
         modelOutput.fRangeValid = false;
         modelOutput.to_from == tofrom_NONE;

         Avionics->bearingD->Caption = "0";
         Avionics->toFromD->Caption = "0";
         Avionics->courseDevD->Caption = "0";
         AnsiString callsign = "";
         callsign += char('A' );
         callsign += char('A' );
         callsign += char('A' );
         Avionics->callsignD->Caption = callsign;
     }

     else
     {
         // refresh the tacan mode value of a/c
         ownship.tacanMode = locals.mode;

         // update bearing value from model output.
        Avionics->bearingD->Caption=IntToStr(modelOutput.bearing);
         // update callsign from model output.
         AnsiString callsign = "";

              callsign+=char('A'+lockedStation.channel.callsign.le
              tter1-1);

              callsign+=char('A'+lockedStation.channel.callsign.le
              tter2-1);

              callsign+=char('A'+lockedStation.channel.callsign.le
              tter3-1);
         Avionics->callsignD->Caption = callsign;
         // update to/from value
         if( modelOutput.to_from == tofrom_FROM )
           Avionics->toFromD->Caption = "FROM";
         else if( modelOutput.to_from == tofrom_TO )
```

```
          Avionics->toFromD->Caption = "TO";
        // update CDI value.
        Avionics->courseDevD-
              >Caption=IntToStr(modelOutput.courseDeviation );
    }
//  update  range  value  from  model  output,  considering  range
//valid signal
    if( modelOutput.fRangeValid )
    {
      Avionics->rangeD->Color = clLime;
      Avionics->rangeD->Caption = IntToStr( modelOutput.range );
    }
    else
    {
      Avionics->rangeD->Color = clRed;
      Avionics->rangeD->Caption = "XXXX";
    }

    // update test LED according to IBIT status.
    if( modelOutput.testLED )
          Avionics->testLedD->Color = clLime;

    else
          Avionics->testLedD->Color = clGreen;

}

void C_TACAN_Model::processInputs(){

    processModeSwitch();
    processIbitRequest();
    ownship.channel.no = modelInput.channelNo;
    ownship.channel.band = modelInput.bandSwitch;
    ownship.position.latitude = modelInput.ACLatitude;
    ownship.position.longitude = modelInput.ACLongitude;
    ownship.position.altitude = modelInput.ACAltitude;
}

void C_TACAN_Model::processIbitRequest()
{
      if( modelInput.ibitRequest )
      {
            locals.ibitTimer = C_IBIT_TIME;
            modelInput.ibitRequest = false;
      }

}

void C_TACAN_Model::requestIBIT()
{
      modelInput.ibitRequest = true;
}

void C_TACAN_Model::processModeSwitch()
{
      if( locals.mode == mode_OFF )
      locals.mode = modelInput.modeSwitch;

      else
```

```
      {
         if( locals.prevMode == mode_OFF )
            locals.warmupTimer = C_TCN_WARMUP_TIME;
         else
            locals.mode = modelInput.modeSwitch;
      }
      locals.prevMode = modelInput.modeSwitch;
}

/*A/A mode channel operation assumed as same Ground operation.
 Simply the channel value is checked with ownship channel value.
 Finds the first matching station, even if other matching stations
exist.Assigns it to lockedStation variable. */

bool C_TACAN_Model::findMatchingStation()
{
    Tacan_Type selectedType;
    if (Avionics->startButton->Caption == "STOP")
    {
        switch(ownship.channel.no){
              case 10: {
                lockedStation.position.latitude=38,15;
                lockedStation.position.longitude=29,50;
                lockedStation.position.altitude=2550;
                lockedStation.channel.callsign.letter1='A';
                lockedStation.channel.callsign.letter2='N';
                lockedStation.channel.callsign.letter3='K';
                lockedStation.type=tacan_ground;
                break;

              }
              case 20: {
                lockedStation.position.latitude=40,05;
                lockedStation.position.longitude=29,50;
                lockedStation.position.altitude=2000;
                lockedStation.channel.callsign.letter1='E';
                lockedStation.channel.callsign.letter2='S';
                lockedStation.channel.callsign.letter3='K';
                lockedStation.type=tacan_ground;
                break;

             }
             case 30: {
                 lockedStation.position.latitude=36,10;
                 lockedStation.position.longitude=30,35;
                 lockedStation.position.altitude=1000;
                 lockedStation.channel.callsign.letter1='C';
                 lockedStation.channel.callsign.letter2='N';
                 lockedStation.channel.callsign.letter3='K';
                 lockedStation.type=tacan_ground;
                 break;

            }
            case 40: {
                 lockedStation.position.latitude=39,10;
                 lockedStation.position.longitude=32,45;
                 lockedStation.position.altitude=500;
                 lockedStation.channel.callsign.letter1='A';
                 lockedStation.channel.callsign.letter2='I';
```

```
                lockedStation.channel.callsign.letter3='R';
                lockedStation.type=tacan_air;
                break;

            }
            default:
                lockedStation.type = tacan_none;
                modelOutput.fRangeValid = false;
                modelOutput.to_from == tofrom_NONE;
            }
    }
    else
      lockedStation.type = tacan_none;
      modelOutput.fRangeValid = false;

      switch( ownship.tacanMode ){
        case mode_AA_REC:
        case mode_AA_T_R:
            selectedType = tacan_air;
            break;
        case mode_REC:
        case mode_T_R:
            selectedType = tacan_ground;
            break;
        default:
            locals.errorCode = ECODE_UNEXPECTED_MODE;
            selectedType = tacan_none;
      }
}

bool C_TACAN_Model::isInStationBlindZone()
{

    bool res = true;
    double deltaZ;
    double elevation;
    deltaZ          =          ownship.position.altitude          -
              lockedStation.position.altitude;
    if( locals.range != 0.0 )
    {
        elevation=180.0 * asin( deltaZ / locals.range ) / M_PI;
        if( elevation < C_TACAN_BLIND_ZONE_ELEVATION )
            res = false;
        else
            res = true;
    }
    else
      res = false;

   return res;
}
bool C_TACAN_Model::isStationInOwnshipDetectionRange()
{
      bool res = false;
      if( locals.range < C_OWNSHIP_DETECTION_RANGE )
            res = true;
      return res;
}
```

```
void C_TACAN_Model::ibitCycle()
{
     if( locals.ibitTimer > C_TEST_0_TIME )
     {
            modelOutput.testLED = true;
     }
     else if( locals.ibitTimer > C_TEST_1_TIME )
     {
            modelOutput.testLED = false;
            modelOutput.bearing = 270;
            modelOutput.fRangeValid = true;
            modelOutput.range = 0;
     }
     else if( locals.ibitTimer > C_TEST_2_TIME )
     {
            modelOutput.bearing = 180;
            modelOutput.fRangeValid = false;
     }
     locals.ibitTimer--;
}

void C_TACAN_Model::tacanCalculate()
{
    findMatchingStation();
    if( lockedStation.type != tacan_none )
    {
      locals.range = calculateRange();
      locals.bearing = calculateBearing();

      if( true == isStationInOwnshipDetectionRange() )
      {
            if( false == isInStationBlindZone() )
            {
                  switch( locals.mode ){
                    case mode_REC:
                        locals.fBearingValid = true;
                        locals.fRangeValid = false;
                        break;

                    case mode_T_R:
                        locals.fBearingValid = true;
                        locals.fRangeValid = true;
                        break;

                    case mode_AA_REC:
                        locals.fBearingValid = false;
                        locals.fRangeValid = false;
                        break;

                    case mode_AA_T_R:
                        locals.fBearingValid = false;
                        locals.fRangeValid = true;
                        break;

                    default:
                        locals.errorCode = ECODE_INVALID_MODE;
                  }
                  if( locals.fBearingValid )
                  {
```

```
                    modelOutput.bearing = (int) locals.bearing;
                    modelOutput.courseDeviation=
                        calculateDeviation();
                }
                else
                {
                  modelOutput.courseDeviation = 0;
                  modelOutput.bearing = invalidBearingValue();
                }

                modelOutput.to_from = calculateToFrom();

                if( locals.fRangeValid ){
                  modelOutput.fRangeValid = true;
                  modelOutput.range = (int) locals.range;
                }
            }
        }
    }
}

double C_TACAN_Model::calculateBearing(){

    int res;
    double bearing;
    Vector_Type p0, p1, range;
    p1.y = lockedStation.position.latitude;
    p1.x = lockedStation.position.longitude;
    p0.y = ownship.position.latitude;
    p0.x = ownship.position.longitude;
    range = deltaPPCalc( p0, p1 );
    bearing = 90.0 - 180.0 * atan2( range.y, range.x ) / M_PI;

    if( bearing >= 360.0 )
      bearing -= 360.0;
    else if( bearing < 0.0 )
      bearing += 360.0;
    res = bearing;
    return res;
}

int C_TACAN_Model::calculateDeviation(){

      int res;
      res = (int) ( locals.bearing - modelInput.course);
      return res;
}

double C_TACAN_Model::calculateRange(){
    double res;
    Vector_Type p0, p1, range;
    double deltaZ;
    double absRange;

    p0.y = lockedStation.position.latitude;
    p0.x = lockedStation.position.longitude;
    p1.y = ownship.position.latitude;
    p1.x = ownship.position.longitude;
```

```
    locals.deltaZ      =      lockedStation.position.altitude      -
                    ownship.position.altitude;
    range = deltaPPCalc( p0, p1 );
    absRange  =  sqrt(  range.x  *  range.x  +  range.y  *  range.y  +
              deltaZ * deltaZ );

    res = absRange;
    return res;
}

//-------------------------------------------------------------
Vector_Type   &C_TACAN_Model::deltaPPCalc   (   Vector_Type   pos0,
Vector_Type pos1 )
{
    static Vector_Type result;
    double cnexz1, cnexx1;
    double polar_radius, equatoral_radius;
    double lat0, long0, lat1, long1;

    lat0 = pos0.y;
    long0 = pos0.x;
    lat1 = pos1.y;
    long1 = pos1.x;

    cnexz1 = sin(lat0 * M_PI / 180.0);
    cnexx1 = cos(lat0 * M_PI / 180.0);

     polar_radius=R_EARTH*(1.-2.*ECC_EARTH+
          3.*ECC_EARTH*cnexz1*cnexz1);
     equatoral_radius=R_EARTH*(1.+
          ECC_EARTH*cnexz1*cnexz1)*cnexx1;
    result.y = (lat1 - lat0) * polar_radius * M_PI / 180.0;
    result.x  = (long1 - long0)* equatoral_radius * M_PI / 180.0;

    return result;
}

Position_Type &C_TACAN_Model::GetStationPosition(unsigned int no){

      return stations[no].position;
}

void C_TACAN_Model::SetOwnshipPosition( Position_Type pos ){

   ownship.orientation.heading=StrToInt(Avionics->HeadingAcEn-
         >Text);
   ownship.orientation.pitch = 0.0;
   ownship.orientation.roll = 0.0;
   ownship.position = pos;
}
```