E. Ü. FEN BİLİMLERİ ENSTİTÜSÜ

EGE ÜNİVERSİTESİ

YÜKSEK LİSANS TEZİ

AKILLI UÇUŞ YÖNETİM SİSTEMİ TASARIMI ÜZERİNE

Harun BAVUNOĞLU

Tez Danışmanı : Prof. Dr. Urfat NURİYEV

Matematik Anabilim Dalı

Bilim Dalı Kodu : 619.03.03
Sunuş Tarihi : 10.06.2011

Bornova-İZMİR
2011

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

AKILLI UÇUŞ YÖNETİM SİSTEMİ TASARIMI ÜZERİNE

Harun BAVUNOĞLU

Tez Danışmanı : Prof. Dr. Urfat NURİYEV

Matematik Anabilim Dalı

Bilim Dalı Kodu : 619.03.03
Sunuş Tarihi : 10.06.2011

Bornova-İZMİR

2011

Sayın Harun BAVUNOĞLU tarafından YÜKSEK LİSANS TEZİ tezi olarak sunulan “Akıllı Uçuş Yönetim Sistemi Tasarımı Üzerine” başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 10.06.2011 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: .....**	**.....**
Raportör Üye	**: .....**	**.....**
Üye	**: .....**	**.....**

ÖZET

AKILLI UÇUŞ YÖNETİM SİSTEMİ TASARIMI ÜZERİNE

BAVUNOĞLU, Harun

Yüksek Lisans Tezi, Matematik Bölümü
Tez Danışmanı: Prof. Dr. Urfat NURİYEV
Haziran 2011, 37 sayfa

Bu tezde, okyanus aşırı uçan uçaklar için iki farklı tür acil durum anında gerçekçi ve akıllı acil iniş rota planlaması yapan sistem oluşturulması amaçlanmıştır.

Bu türlerden ilki havalimanına en kısa zamanda iniş gerektiren durumlar ki bunlar bir yolcunun kalp krizi geçirmesi, uçağın yapısalındaki herhangi bir hasar, kabin basıncının aniden düşmesi, yangın çıkması vb. gibi durumlar olabilir bu durumlarda amaç, uçağın yere göre hızını uçağın limitlerine göre en üst seviyede tutup acil iniş noktasına en kısa sürede varmasını sağlamaktır. İkinci tür acil durum olarak uçağın motorlarında yakıt sızıntısı olması ele alınmıştır ve bu durumlarda amaç, zaman kısıtı olmadan uçağın en az yakıtı harcayarak yani rüzgarın hızını olabildiğince etkin kullanarak acil iniş noktasına varmasını sağlamaktır.

Rotanın bulmasında sezgisel arama yöntemi olan, A yıldız(A*) arama algoritması kullanılmıştır. Rotanın bulunma süresinin kabul edilebilir olması için kullanılan veri yapıları ve algoritmalar optimize edilmiştir.

Gerçeğe olabildiğince yakın veriler kullanılmış, hesaplamalardaki detaylardan kaçınılmamış ve optimum rota, kabul edilebilir bir sürede bulunmuştur.

Anahtar sözcükler: Uçuş rota planlaması, acil durum rotası, A* algoritması, A yıldız algoritması

ABSTRACT

ON INTELLIGENT FLIGHT MANAGEMENT SYSTEM DESIGN

BAVUNOĞLU, Harun

MSc in Mathematics
Supervisor: Prof. Dr. Urfat NURİYEV
June 2011, 37 pages

In this thesis, its aimed to build a system which makes realistic and smart route planning for emergency landing for transoceanic aircrafts in two kind of emergency situation.

One of that kind of emergency situation is that the stiuation which is required soonest landing. Examples of this kind of stiuation may be a passenger having heart attack, any damage of structure of aircraft, sudden and rapid depressurization of the aircraft cabin, fire on board the aircraft etc. The aim is keeping the ground speed on upper limit of aircraft and guaranteeing the landing soonest as possible in this kind of situation. The other kind of emergency situation is that the situation which is the aircraft engine has fuel leakage. The aim is keeping the fuel flow minimum by using the advantages of wind without any time constraints.

A star(A*) algorithm which is heuristic search method is used by route finding. Used data structures and algorithms are optimized to make the time of finding emergency route acceptable.

As near as possible to the reality datas were used, did not avoid detailed calculations and optimum route was found in acceptable amount of time.

Keywords: Flight planning, Emergency landing, A* algorithm, A star algorithm

TEŞEKKÜR

Bu çalışma süresince, özellikle kıymetli görüşlerinden yararlandığım ve tezin biçimlenmesinde değerli katkılarını aldığım danışman hocam sayın Prof. Dr. Urfat Nuriyev'e, Güray Yıldız ve Elif Saygı' ya, gerekli verilerin sağlanmasında kolaylık gösteren TUSAŞ-Türk Havacılık ve Uzay Sanayii A.Ş.' ne ve çalışanlarına teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ÖZET v

ABSTRACT vii

TEŞEKKÜR ix

ŞEKİLLER DİZİNİ xii

ÇİZELGELER DİZİNİ xiii

SİMGELER VE KISALTMALAR DİZİNİ xiv

1. GİRİŞ 1

2. SİSTEMİN AKIŞI 5

2.1 Mevcut Grid Numarasının Bulunması 5

2.2 Hedef Grdilerin Aranması 6

2.3 A Yıldız(A*) Algoritması 9

2.3.1 A* algoritması 10

2.3.2 Sezgisel fonksiyonun muteberlik ispatı 15

2.3.3 Maliyet hesaplaması 16

2.4 Optimalleştirme 23

İÇİNDEKİLER (devam)

Sayfa

3. DENEYSEL BULGULAR.....28

3.1 Deney Koşulları.....28

3.2 Deney Sonuçları 30

4. SONUÇ.....31

5. ÖNERİLER 32

EK AÇIKLAMALAR 33

KAYNAKLAR DİZİNİ.....34

ÖZGEÇMİŞ.....37

EKLER .....

Ek 1 Kaynak Kod .....

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

1.1 Büyük Okyanus Hava Durumu 2

2.3a AutoDesk-Kynogon..... 9

2.3b Robot A* algoritmasını çalıştırıyor..... 10

2.3.1.1 A* algoritması koşturulması sonucu adım 1..... 12

2.3.1.2 A* algoritması koşturulması sonucu adım 2..... 13

2.3.1.3 A* algoritması koşturulması sonucu adım 3..... 13

2.3.1.4 A* algoritması koşturulması 14

2.3.1.5 A* algoritması optimum yolu buluyor..... 15

2.4.1 İkili En Küçük Yığını..... 23

2.4.2 İEKY'nin Dizideki Görünüşü 24

2.4.3 Düğüm Ekleme..... 24

2.4.4 Düğüm Çıkarma..... 25

2.4.5 Yığın veri yapısının bağlı liste gösterimi..... 27

3.1.1 Uçuş Alanı.....28

3.2.1 Simülatör.....30

ÇİZELGELER DİZİNİ

Çizelge **Sayfa**

2.1.1 Grid Numaraları 5

2.2.1 Algoritmanın çalışma örüntüsü 7

2.3a En kısa yol algoritmalarının zamansal karşılaştırması 9

2.3.3.2.1 Uçağın derece cinsinden yön bilgisi 21

2.4.1 İEKY yapısının karmaşıklık tablosu 26

3.1.2 Hava Sıcaklığının Yakıt Akışına Etkisi.....29

3.2.1 Deney Sonuçları.....30

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar

İHA	İnsanız Hava Aracı
GPS	Global Pozisyon Sistemi
VOR	Çok Yüksek Frekanslı 360 Derece Yönlü Uzaklık
DME	Mesafe Ölçme Ekipmanı
TACAN	Taktiksel Hava Navigasyon Sistemi
ADC	Hava Veri Bilgisayarı
NM	Deniz Mili
FMS	Uçuş Yönetim Sistemi

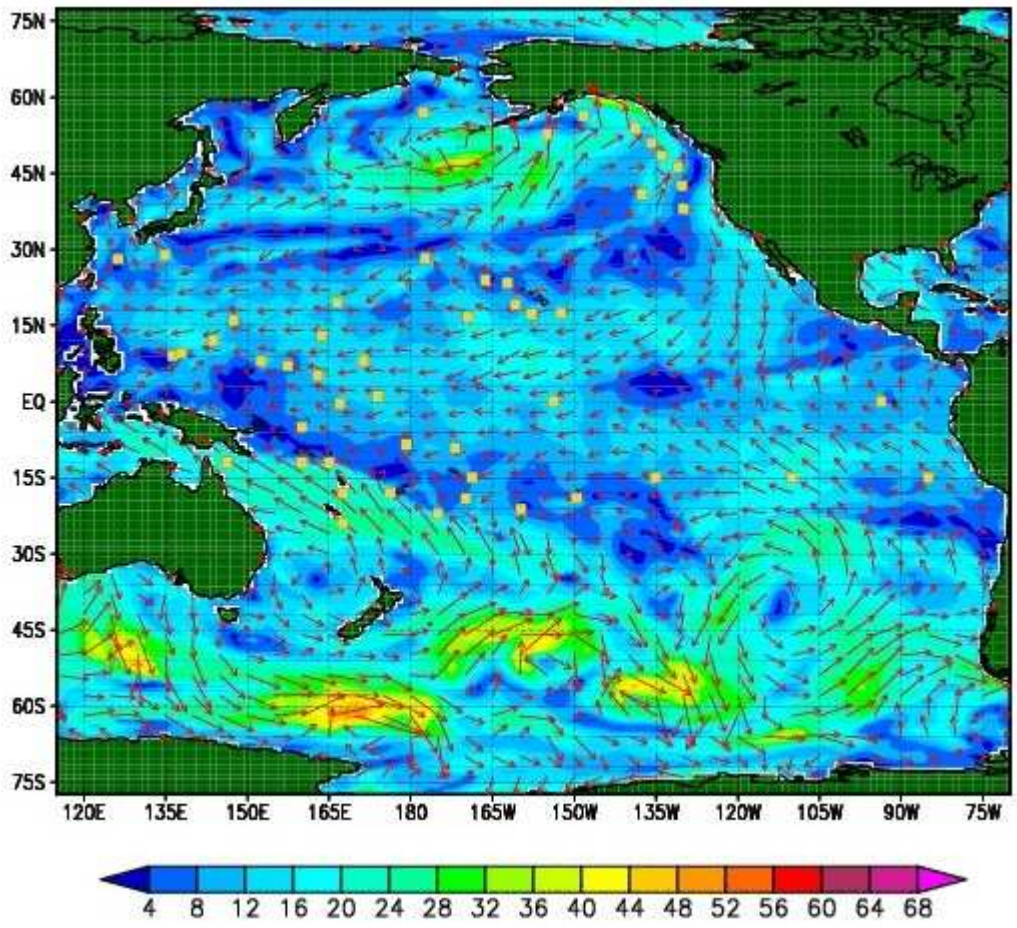
1. GİRİŞ

19 Temmuz 1989’ da Denver’ dan Chicago’ ya giden uçağın arka iki motorunun da arızalanması sonucu pilotlar, uçağın kontrolünü kalan iki motorla mucizevi bir şekilde sağlayıp Sioux City’ ye acil iniş yapmak durumunda kalmışlardır. 24 Ağustos 2001’ de, Toronto’ dan Lizbon’ a olan Air Transat firmasının 236 sefer sayılı uçuşunda pilotların fark ettiği yakıt sızıntısı herhangi bir can veya mal kaybı olmadan, fakat normal iniş hızından iki kat fazla olduğu için küçük yaralanmalarla atlatılmıştır. 08 Ocak 2011’ de Amerikan F16’ sı İskoçya’ ya yakıt sızıntısından dolayı acil iniş yapmak zorunda kalmıştır. 22 Şubat 2010’ da Thomas Cook firmasına ait bir yolcu uçağı yakıt sızıntısı sebebiyle acil iniş yapmak zorunda kalmıştır. Uçuş sırasında herhangi bir acil durumun bütün yükünü pilotlar üstlenmek zorundadırlar, bu yükün biraz olsun hafifletilmesi pilotların sağlıklı karar verebilmelerini kolaylaştırmaktadır. Bu kapsamda özellikle okyanus aşırı uçuşlarda oluşabilecek acil durum türlerine özgü, hava koşullarının avantajlarını kullanıp, dezavantajlarını olabildiğince ortadan kaldırarak ve uçağın performansını göz önüne alarak en yakın havalimanına üç boyutlu rota planlaması yapılması gerekmektedir.

Bu tezde, okyanus aşırı uçan uçaklar için iki farklı tür acil durum anında gerçekçi ve akıllı acil iniş rota planlaması yapan sistem oluşturulması amaçlanmıştır.

Bu türlerden ilki havalimanına en kısa zamanda iniş gerektiren durumlar ki bunlar bir yolcunun kalp krizi geçirmesi, uçağın yapısalındaki herhangi bir hasar, kabin basıncının aniden düşmesi, yangın çıkması vb. gibi durumlar olabilir bu durumlarda amaç, uçağın yere göre hızını uçağın limitlerine göre en üst seviyede tutup acil iniş noktasına en kısa sürede varmasını sağlamaktır. İkinci tür acil durum olarak uçağın motorlarında yakıt sızıntısı olması ele alınmıştır ve bu durumlarda amaç, zaman kısıtı olmadan uçağın en az yakıtı harcayarak yani rüzgarın hızını olabildiğince etkin kullanarak acil iniş noktasına varmasını sağlamaktır.

Rotanın bulunma sürecinde, hava tahmin verileri, yeryüzü şekilleri, uçağın performans verileri detaylı bir şekilde kullanılmıştır.(UTED, 2011; Wikipedia, 2011) Rotanın bulunma süresinin kabul edilebilir olması için kullanılan veri yapıları ve algoritmalar optimize edilmiştir.



Şekil 1.1 Büyük Okyanus Hava Durumu

Rota planlaması yapılırken uygun rotanın aranması grid bazlı yapılmıştır. Bu seçim hassaslıktan tolere edilebilir ölçüde ödün verip hesaplama zamanından büyük kazanç sağlamıştır.

Kullanıcı dilediği miktarda, tabiki hesaplama zamanını göz önüne alarak alternatif rota oluşturtabilecektir. Bu alternatif rotalar planlananla o anki değerler tutmadığında yine pilotun karar zamanını kısaltma amacıyla oluşturulacaktır.

Rota hesaplamalarında göz önüne alınacak iki girdiden biri hava şartları diğeri uçak performans değerleridir. Hava şartları ki bunlar farklı irtifalar ve zaman periyodları için rüzgarın yönü, rüzgarın hızı ve hava sıcaklığıdır, avantaj sağlayacak şekilde, bunu gerçekleştirirken de uçağın performas girdilerine uygun olması koşulu ile üç boyutlu rota planlaması yapılmıştır.

Bu sistem, hava tahmin girdilerinin olması sebebiyle statik graf arama algoritması kullanılarak tasarlanmıştır. Hava tahmin değerlerinin olmaması ya da tahminlerin tutması durumu öneriler kısmında ele alınmıştır. Statik graf arama algoritmaları arasından A* baz alınmıştır (Hart et al., 1968). Bunun sebeplerinden ilki sezgisel arama olması ve hesaplama zamanını yüksek oranda azaltmasıdır bir diğeri kolay uygulanabilir ve anlaşılabilir olmasıdır.

Sistemin kodlaması sırasında farklı programlama dillerinin avantajları ve veri yapılarının uygunluğu göz önüne alınmıştır. Gerçeğe çok yakın veriler kullanılmış ve olumlu sonuçlar alınmıştır. Mümkün olduğu ölçüde parametrik kodlanarak esneklik ve tekrar kullanılabilirliği azami seviyede tutmak hedeflenmiştir.

Hava şartları havacılıkta rota planlamasındaki en önemli faktörlerden biridir. Bu konuyla ilgili bir çok çalışma yapılmıştır. Özellikle fırtınalardan kaçınılarak rota planlanması yapılması gibi. Ulaşılması gereken havalimanına en kısa uçuş fırtına içinden geçerek gerçekleştirilse bile fırtınadan kaçmak daha güvenli olduğundan pilotu bu rotalara yönlendirmek için çalışmalar yapılmıştır.(Krozel et al., 2006; Meuleau et al., 2008).

Düzlem üzerinde optimum yolu bulmak için yapılan araştırmalar daha sonra uzaya yani üç boyuta taşınmıştır (Stefanakis ve Kavouras, 2002).

Askeri havacılıkta da tekrar rota planlaması oldukça sık kullanılmaktadır, en çok da özel görevlerde kullanılmak üzere tasarlanmış insansız hava araçlarında(İHA). Yer kontrol birimiyle iletişimlerini kaybetmeleri halinde, İHA' nın düşmemesi için otonom bir şekilde kalktığı havalimanına inebilme kabiliyetine sahip olması gerekmektedir (Cook ve Smallman, 2008).

2. SİSTEMİN AKIŞI

2.1 Mevcut Grid Numarasının Bulunması

Bir acil durum oluştuğunda öncelikle üç boyutlu konum bilgisi gerekmektedir. Konum bilgisi iki kısımdan oluşmaktadır biri yatay diğeri dikey kısımdır. Uçuş sırasında bu yatay navigasyon bilgisi eğer varsa FMS ekipmanından yoksa GPS' ten veya VOR, DME, TACAN kulelerinden alınmalıdır.(Ferguson,1999) Bu yatay konum bilgisiyle(Enlem, Boylam) önceden belirlenmiş hassaslıktaki grid dizisinden hangi gridin içinde olduğu formül *(1)* ile bulunur. Bu grid dizisi önceden belirlenmiş uçuş alanından oluşturulur. Gridler yatayda sağa ve yukarı artacak şekilde numaralandırılır. Örneğin 14 x 8' lik bir grid yapısı Çizelge 2.1.1' deki gibi numaralandırılır.

Çizelge 2.1.1 Grid Numaraları

104	105	106	107	108	109	110	111
96	97	98	99	100	101	102	103
88	89	90	91	92	93	94	95
80	81	82	83	84	85	86	87
72	73	74	75	76	77	78	79
64	65	66	67	68	69	70	71
56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

$X_c Y_c$ enlem, boylam olarak alınan yatay konum bilgisi,

Güneybatı noktası $X_1 Y_1$ ve kuzeydoğu noktası $X_2 Y_2$ olan m x n' lik grid yapısı,

g hassasiyet olmak üzere;

$$n = \left\lfloor \frac{Y_2 - Y_1}{g} \right\rfloor$$

$$\text{GridNo} = \left(n * \left\lfloor \frac{X_c - X_1}{g} \right\rfloor \right) + \left\lfloor \frac{Y_c - Y_1}{g} \right\rfloor \qquad (1)$$

Diğer kısım olan dikey konum bilgisi ise ADC ya da Altimetreden alınmalıdır. Uçaklar için ortalama maksimum irtifa deniz seviyesinden 60.000 feet' e (10 NM = 18,5km) kadar olarak alınmış ve dikey olarak 10 eşit irtifa aralığına bölünmüştür. Alınan irtifanın hangi irtifa aralığına düştüğü formül *(2)* ile bulunur.

A_c alınan irtifa bilgisi olmak üzere,

$$\text{ANo} = \lfloor A_c \rfloor \qquad (2)$$

2.2 Hedef Gridlerin Aranması

Uçak grid yapısına göre konumlandırıldıktan sonra yani, rota planlamak için başlangıç gridi belirlendikten sonra önceden sisteme girdi olarak sağlanmış sayıda en yakın hedef grid, yani havalimanı belirlenmelidir. Bu hedef gridler irtifa aralıklarının sadece ilkinde yani yere en yakın olanında bulunabilirler. Bu da hedef grid arama uzayını düzleme indirgemiştir. Tüm gridleri taramamak ve hesaplama zamanını kısaltmak için önceden belirlenmiş sayıdaki hedef gride ulaşıncaya kadar başlangıç gridinden spiral çizerek taranan grid sayısı genişletilmektedir. Bu arama, en yakın

istenilen sayıda hedef grid bulacağını garanti eder ve en kötü durumda karmaşıklığı O(n)' dir.

Örneğin 14x8' lik bir grid yapısında 4 en yakın hedef grid için arama örüntümüz Çizelge 2.2.1' deki gibi olacaktır.

Çizelge 2.2.1 Algoritmanın çalışma örüntüsü

104	105	106	107	108	109	110	111
96	97	98	99	100	101	102	103
G 88	89	90	91	92	93	94	95
80	81	82	83	84	85	86	87
72	73	74	75	76	77	G 78	79
64	65	66	67	68	69	70	71
56	57	58	S 59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	G 47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	G 17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

m x n grid yapısı,

l önceden bilirlenmiş havaalanı sayısı kadar eleman tutan liste,

s başlangıç grid numarası olmak üzere;

Adım 1) fark = 1

Adım 2) s = s + n , s<= m * n ise s' te havaalanı olup olmadığına bak varsa l' ye at,

l dolduysa dur.

Adım 3) fark kadar dön

Adım 4) s = s + 1 , (s mod n) 0’ a eşit değilse s’ te havaalanı olup olmadığına bak varsa l’ ye at, l dolduysa dur.

Adım 5) fark + 1 kadar dön

Adım 6) s = s – n, s>= 0 ve (s mod n) 0’ a eşit değilse s’ te havaalanı olup olmadığına bak varsa l’ ye at, l dolduysa dur.

Adım 7) fark + 1 kadar dön

Adım 8) s = s – 1, ((s+1) mod n)0’ a eşit değilse s’ te havaalanı olup olmadığına bak varsa l’ ye at, l dolduysa dur.

Adım 9) fark + 1 kadar dön

Adım 10) s = s + n ve s’ te havaalanı olup olmadığına bak varsa l’ ye at, l dolduysa dur.

Adım 11)fark = fark + 2 ve {2} ye dön

Acil iniş için havalimanlarının içinde olduğu gridler bulunduktan sonra başlangıç gridinden bu gridlere rota planlaması yapılmaktadır. Rota planlaması yapılırken A* algoritması temel alınmış, acil durum türüne göre maliyet fonksiyonları belirlenmiştir.

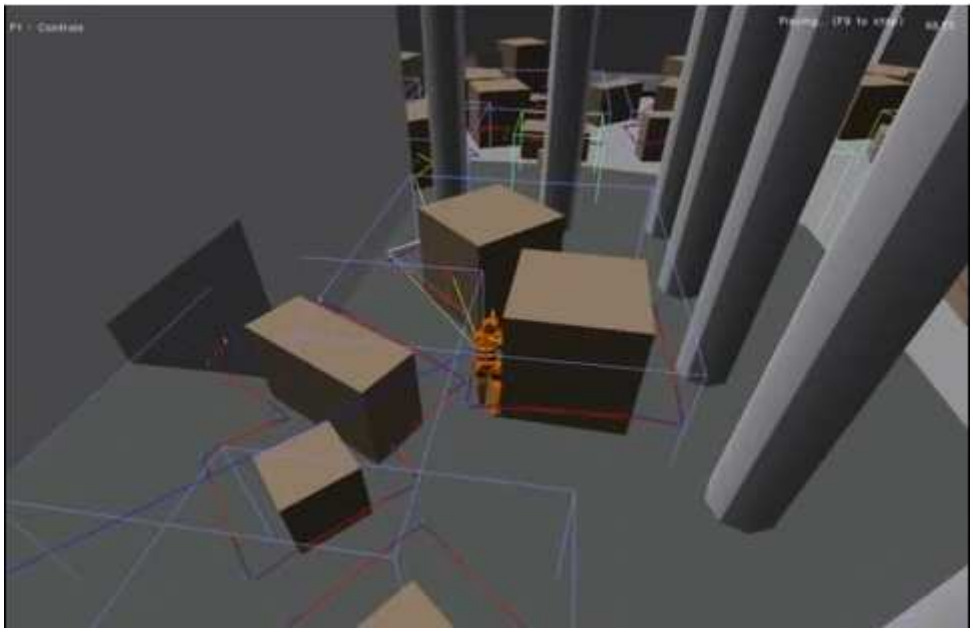
2.3 A Yıldız(A*) Algoritması

Graf arama algoritmalarının hesaplama süreleri, grafın boyutuna göre çok uzun olabilir. Hesaplama sürelerini kabul edilebilir seviyelere çekmek için sezgisel arama yöntemleri geliştirilmiştir.**(**Russell, 2003; Nabiyev, 2005; Nabiyev, 2007)

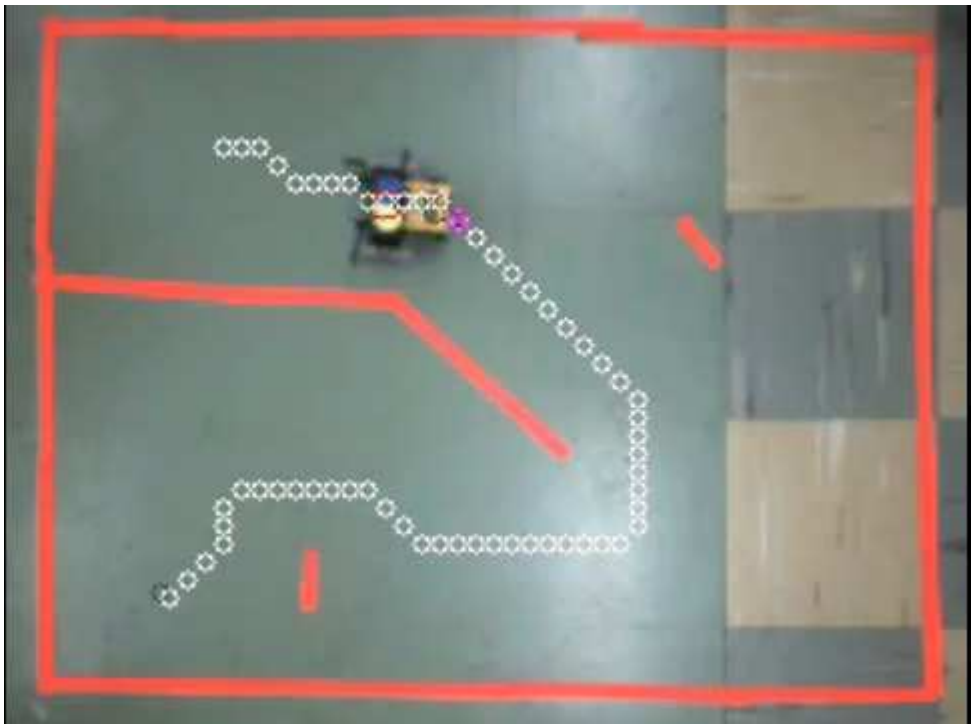
Çizelge 2.3a En Kısa Yol Algoritmaların Zamansal Karşılaştırması(Santoso et al., 2010)

Algoritma	Çalışma Zamanı(3007,61m için)
Dijkstra	140 ms
A*(Sezgisel)	**94 ms**
Karınca Kolonisi	13604 ms

Bunlardan en popüleri A* algoritmasıdır. Oyunlar, robot uygulamaları gibi birçok uygulama alanı vardır.



Şekil 2.3a AutoDesk-Kynogon



Şekil 2.3b Robot A* algoritması çalıştırıyor

A* algoritması, seçilen sezgisel fonksiyonu muteber sezgisel olduğunda optimum rotayı garanti eder. Öncelikle A* algoritması anlatılacak ve sonrasında seçilen sezgiselin muteberliği ispatlanacaktır.

2.3.1 A* algoritması

Açık Liste: Hesaplamaları yapılmış fakat seçilmemiş gridlerin listesi

Kapalı Liste: Hesaplamadan sonra seçilmiş gridlerin listesi

g(n) : Başlangıç gridinden n. gride gelişin toplam maliyeti,

h(n) : n. gridden hedef gride gidişin sezgisel maliyeti,

$$f(n) = g(n) + h(n)$$

olmak üzere;

Adım 1) Başlangıç gridini açık listeye ekle

Adım 2) Aşağıdakileri tekrar et :

a) Açık listedeki en küçük F değerine sahip olan gridi al ve mevcut grid olarak ata

b) Açık listeden silip kapalı listeye ekle

c) Bütün 26 komşu gridler için (üç boyutta mevcut grid hariç)

- Eğer komşu grid kapalı listede veya gidilemez ise diğer komşuya geç, değilse aşağıdakileri yap.
  - Eğer komşu açık listede değilse açık listeye ekle, mevcut gridi kendisine ebeveyn grid olarak ata. F ,G ve H değerlerini hesapla.
  - Eğer açık listede ise G değerini kullanarak o yolun daha iyi olup olmamasına bak. G değeri düşük olan daha iyi bir yola sahip demektir. F ve G değerini yeniden hesapla. Eğer açık liste F değerine göre sıralı değil ise tekrar sırala.

d) Aşağıdakilerden biri sağlandığı zaman dur:

- Hedef gridi kapalı listeye ekle, bu noktada artık bir rota bulunmuş oluyor.
- Açık listenin boş olması, yani hedef gridin bulunamaması durumunu

Adım 3) Bulunan rotayı kaydet ve hedef gridden başlangıç gridine bulana kadar geriye doğru ebeveyn gridlerini takip et. Bulunan yolu döndür.

Kolay anlaşılması için iki boyutlu örnek üzerinde incelenirse;

Yeşil grid başlangıç, kırmızı grid hedef, mavi gridler de gidilemeyen gridler olsun.

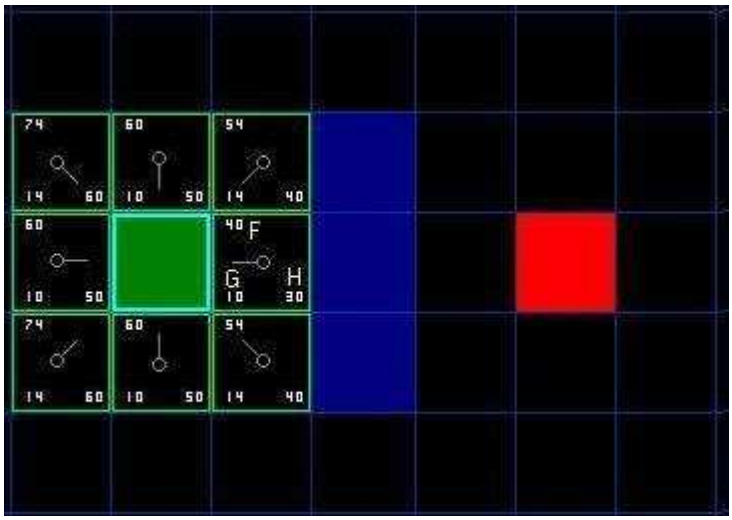
Bir gridden diğerine gitme maliyeti diyagonaldekiler için

$$c(a,b) = 10 * \sqrt{2} \cong 14$$

yatay ve düşeydekiler için 10 alınmıştır.

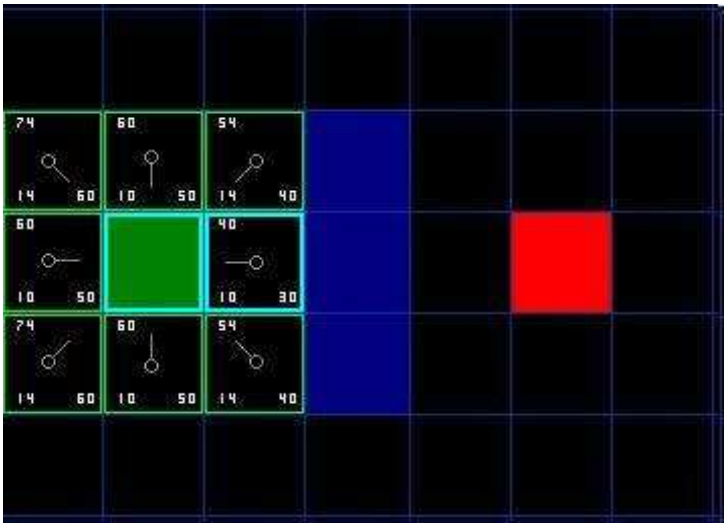
Bir gridden hedef gride sezgisel uzaklık için de maliyet fonksiyonunun kuş bakışı Manhattan uzaklığı alınmıştır.(Policyalmanac, 2005)

Algoritmayı çalıştırmaya başladığımızda başlangıç gridi açık listeye eklenir sonra açık listeden en küçük F değerine sahip grid açık listeden silinip kapalı listeye eklenir ve tüm komşu gridleri için açık veya kapalı listede olmadıkları için F değeri G ve H değerleri toplamından hesaplanır, ebeveyn grid olarak atanır, açık listeye eklenir ve hedef gride ulaşılamadığı için ya da açık liste boş olmadığı için Adım 2' ye dönülür.(Bkz. Şekil 2.3.1.1)

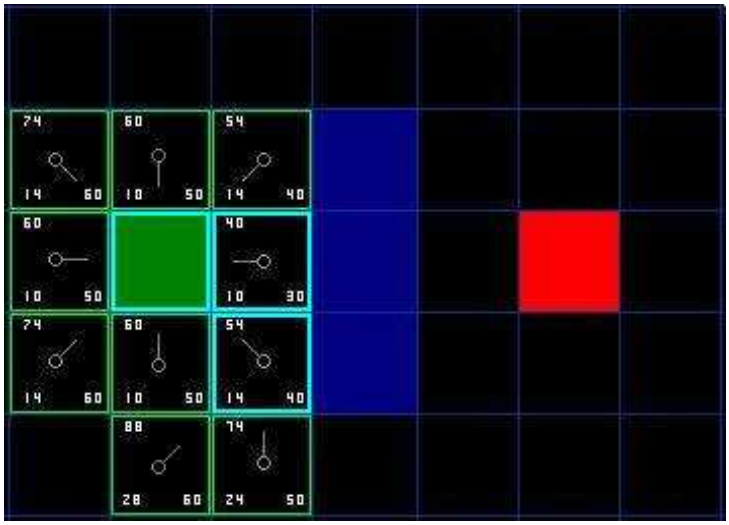


Şekil 2.3.1.1 A* algoritması koşturulması sonucu adım 1

Açık listede en düşük F değeri sahip olan açık listeden silinip kapalı listeye atılır, yani o gride gidilir. Gidilebilen komşu gridlerine bakılır ve açık listede olan komşu grid ile o gridin G değerleri karşılaştırılır 14, 20(10 + 10)’ den daha küçük olduğu için o gridden gitmek daha iyi bir yoldur ve F, G, H değerleri güncellenir. Artık yeni G değerimiz 14, yeni F değerimiz de 54’ tür ve Adım 2’ ye dönülür.

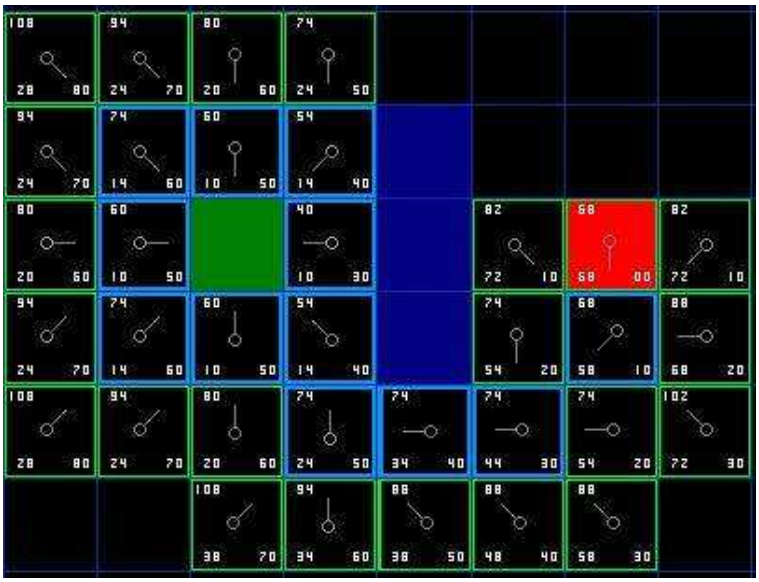


Şekil 2.3.1.2 A* algoritması koşturulması sonucu adım 2



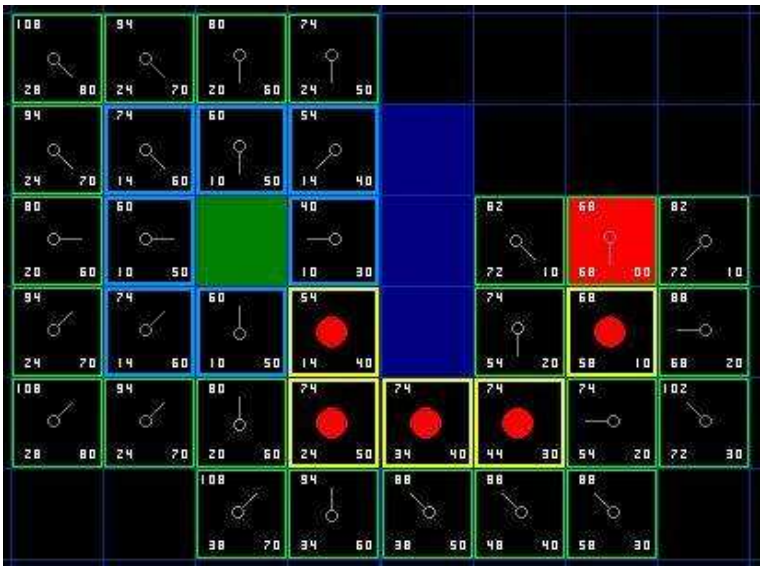
Şekil 2.3.1.3 A* algoritması koşturulması sonucu adım 3

Bu şekilde devam edilerek hedef grid aranır. Gidilen grid yani kapalı listeye atılan grid hedef grid olduğunda algoritma durur ve en kısa yol bulunmuş olur.



Şekil 2.3.1.4 A* algoritması koşturulması sonucu

Bulunan en kısa yolu belirlemek için hedef gridden geriye doğru ebeveynler başlangıç gride ulaşılıncaya kadar takip edilir. Başlangıç gride ulaşıldığında artık en kısa yol belirlenmiş, fazladan gidilenler elenmiş olur.



Şekil 2.3.1.5 A* algoritması optimum yolu belirliyor

Örnek 2-boyutlu olduğu için alınan komşu grid sayısı 8’ dir, fakat bu çalışmada oluşturulacak rota 3-boyutlu olduğu için komşuların sayısı 26 dır, yani (3 x 3 x 3) lük küpün merkezindeki (1 x 1 x 1) lik küp haricindekilerdir.

Bu çalışma kapsamında bir gride gidilememesinin nedenleri, coğrafi engeller, hava şartlarının elverişsizliği, yakıtın bitmesi veya uçağın performans değerlerinin elverişsizliği gibi durumlardır. Maliyet hesapları (G(n), H(n)) kısım 2.3.3’ te detaylı ele alınacaktır.

2.3.2 Sezgisel fonksiyonunun muteberlik ispatı

A* algoritmasının optimum çözümü verdiği sezgisel fonksiyonun muteber sezgisel olması ile ispatlanabilir(Hart et al., 1968). Bu çalışmada kullanılan H(n) sezgisel fonksiyonu için, alınan iki nokta arası en kısa uzaklık hesabının muteberlik olduğu *(3)* de ispatlanmıştır.

$H(n,\ g)$, n gridinden hedef gride(g) hesaplanan sezgisel değer, yani küre üzerindeki n ile g noktaları arasındaki en kısa yay uzunluğu, $C(n,\ g)$, n gridinden hedef gride(g) gidebilmenin hesaplanan gerçek değeri olmak üzere;

$C(n, g) = C(n+1, n+2) + C(n+2, n+3) + \ldots + C(n+x, g)$

şeklinde bulunur.

H(n, g) muteberdir eğer;

$$\forall n, h(n, g) \leq C(n, g) \qquad (3)$$

H(n ,g) sezgiseli büyük çember uzaklığı(Great-Circle Distance) olarak alındığında ve büyük çember uzaklığının da küre üzerindeki en kısa uzaklık olmasından dolayı küre üzerindeki iki nokta arasındaki herhangi başka bir uzaklık büyük çember uzaklığından büyük ya da eşit olacaktır.(IAState Üniversitesi, 2010; Wikipedia, 2011)

2.3.3 Maliyet hesaplaması

2.3.3.1 Rota planlama türü

Bu çalışmada iki tür rota planlama üzerinde durulmuştur. Bunlar, en kısa sürede acil iniş ve en az yakıt tüketimidir. Bu iki tür arasındaki en önemli fark zaman ve yakıt akışı kısıtlarıdır. En kısa sürede iniş türünde, yakıt akışı bakımından herhangi bir kısıt olmazken, en az yakıt tüketimi türünde de herhangi bir zaman kısıtı yoktur. İkisi için de tek kısıt hedef gride ulaşılabilmesi için yeterli yakıtın bulunmasıdır. Bu türlerin matematiksel olarak modellenmesi aşağıdaki gibidir.

En az yakıt tüketimi türü için;

F_r: O anki kalan yakıt miktarı,

F_{min} :İniş için gereken yakıt miktarı,

FF : O anki yakıt akışı,

GS_e: O anki yere göre hız,

GS_{min}: Tutunma kaybı olmaması için gereken yere göre en az hız,

GS_{max}: Yapısal kısımda hasar oluşmaması için gereken yere göre en fazla hız,

Alt_c: O anki irtifa,

Alt_{max}: Çıkılabilecek en yüksek irtifa,

olmak üzere;

$F_r > F_{min}$,

$GS_c \geq GS_{min}$,

$GS_{max} \geq GS_c$,

$Alt_{max} \geq Alt_c$,

kısıtları altında hedef fonksiyon:

$\min(FF)$

en kısa sürede acil iniş türü için aynı kısıtlar altında hedef fonksiyon;

$\max(GS_c)$

olarak belirlenmiştir.

Bu hesaplamalar A* algoritması çalışması sırasında açılan komşu gridlerden hangisine gidilmesinin, o rota planlama türünde daha büyük avantaj sağlayacağını belirlemek için kullanılmaktadır.

2.3.3.2 Koşulların planlamaya etkisi

Rota planlaması yapılırken dikkat edilmesi gereken üç önemli kısıt vardır, bunlar acil durum anının koşulları, hava şartları ve uçağın performans değerleridir. Uçağın performans değerleri;

- Uçağın yapısalının hasar görmemesi için gereken en yüksek hız
- Uçağın yapabileceği en dar dönüş açısı
- İniş için gereken en az yakıt miktarı
- İniş için gereken en kısa uzaklık
- Birim zamandaki yakıt akışı

bilgileridir. Uçağın yapabileceği en dar dönüş açısı ve iniş için gereken en kısa uzaklık grid boyutlarını belirlerken kullanılmaktadır. Acil durumun oluştuğu şartların da maliyet hesabında kullanılması gerekir, bu şartlar;

- Uçaktaki yakıt miktarı
- Uçağın enlem ve boylam olarak pozisyonu, irtifası
- Uçağın hızı
- Uçağın toplam ağırlığı

bilgileridir. Hava şartları rota planlamasında büyük rol oynamaktadır. Özellikle yakıt sızıntısı sırasında rüzgarın avantajı planlama için çok önemlidir. Hava şartları değişkenleri;

- Acil durum anında rüzgarın yönü ve hızı,
- Acil durum irtifasındaki hava sıcaklığı
- Acil durum anının hava tahmin verileri

- Farklı zaman periyotları ve farklı bölgeler için hava tahmin verileri
- Hava radarı görüntüsü değerleri

bilgileridir. Hava radarı görünütüsü değerleri, o anki hava tahmin verileriyle karşılaştırılarak iki değer arasında bir kalibrasyon yapılmaktadır.

Önceki kısım kısımda belirtildiği gibi acil durum türlerine göre farklı maliyet hesapları kullanılmaktadır. Bunlar şu şekilde modellenmiştir;

en kısa sürede acil iniş türü için maliyet değeri,

$C_1(c,d)$: c ve d noktaları arasındaki maliyet değeri,

$\gcd(c,d)$: c ve d noktaları arasındaki büyük çember uzaklığı,

GS : Yere göre hız

olmak üzere;

$$C_1(c,d) = \frac{gcd(c,d)}{GS}$$

şeklinde hesaplanır ve en az yakıt tüketimi türü için maliyet değeri,

$C_2(c,d)$: c ve d noktaları arasındaki maliyet değeri,

$\gcd(c,d)$: c ve d noktaları arasındaki büyük çember uzaklığı,

GS : Yere göre hız,

$FF(c,d)$: c ve d noktaları arasındaki yakıt akış değeri

olmak üzere;

$$C_2(c,d) = FF(c,d) * \frac{\gcd(c,d)}{Gs}$$

şeklinde hesaplanır. Bu hesaplamalar için gerekli olan büyük çember uzaklığı aşağıdaki şekilde hesaplanır;

X_1, Y_1 *ve* X_2, Y_2 : Küre üzerindeki iki noktanın enlem ve boylamları,

D : Bu iki nokta arasındaki büyük çember uzaklığı

olmak üzere;

$$D = \cos^{-1}(\sin(X_1) * \sin(X_2) + \cos(X_1) * \cos(X_2) * \cos(Y_2 - Y_1))$$

şeklinde hesaplanır.(Wikipedia, 2011)

Maliyet hesaplamarı için gerekli bir diğer parametre de yere göre hız değeridir. Yere göre hız, uçağın pito tüpünden ölçtüğü hız ile rüzgarın hızının vektörel bileşkesinden hesaplanır. Bu hesaplamayı yapabilmek için öncelikle uçağın coğrafi kuzeye göre yön bilgisinin bulunması gerekmektedir. Bu bilgi de şu şekilde bulunur; bir gridin üç boyutta 26 komşu gridi vardır fakat yatayda üst 9 ve alt 9 komşu grid için yön değişmeyeceğinden aynı irtifadaki 8 koşmu grid için yön hesaplaması yapılmıştır. Iki grid arasındaki yönü coğrafi kuzeye göre bulmak için aşağıdaki Çizelge 2.3.3.2.1 oluşturulmuştur.

m x n grid yapısı olmak üzere,

Çizelge 2.3.3.2.1 Uçağın derece cinsinden yön bilgisi

Gidilecek grid ile aradaki gridNo farkı	Yön (Derece)
n	0
n + 1	45
1	90
1 - n	135
-n	180
-(n + 1)	225
-1	270
n - 1	315

Iki grid arasındaki yön (artık buna grid yönü denecek) bulunduktan sonra rüzgarın bu yönde etkisi hesaplanır Rüzgarın ektisini katmadan önce hangi zaman periyodu için rüzgar durumuna bakılacağı hesaplanmalıdır, bu da iki grid arası büyük çember uzaklığının yarısının yere göre hıza bölünmesiyle elde edilir. Büyük çember uzaklığının yarısının alınmasının sebebi, iki grid arası işlem yapılırken orta noktaları baz alınır, bu orta noktardan geçerken iki gridin hava durumunun göz önüne alınması gerekir, büyük çember uzaklığının yarısı geçilen gridin verilerine ulaşmak için kullanılır.. Rüzgarın yönüyle gridin yönünün farkı alınarak rüzgar yönü ile grid yönünün arasındaki açı bulunur. Bu açı rüzgarın hızının grid yönünde ne kadarlık hıza karşılık geldiğinin hesaplanmasında kullanılır. Rüzgar hızının grid yönündeki hızıyla o anki hızın toplam değerine yere göre hız denir. Yere göre hız aşağıdaki şekilde hesaplanır. (Aeronautical Inc, 2006; Airbus, 2011)

Eğer uçak tırmanışta ise $TAS_{Climb} = \frac{TAS}{\sqrt{2}}$ şeklinde alınabilir ve eğer rüzgarın yönü ile gridin yönü arasındaki açı 90°, den büyükse yavaşlatıcı, küçükse arttırıcı ektisi olmaktadır. **(**Cary, 2001)

GS : Yere göre hız değeri,

TAS : Pito tüpünden okunan hızın gerçek hava hızına çevrilmiş hali,

WS : Rüzgarın hızı,

α_w : Rüzgarın coğrafi kuzeye göre saat yönünde yönü,

α_c : Uçağın coğrafi kuzeye göre saat yönünde yönü

olmak üzere yere göre hız eğeri;

$$GS = TAS - WS * \cos(2 * \pi - (\alpha_w - \alpha_c)) \text{ , eğer } 2 * \pi - (\alpha_w - \alpha_c) \geq 90°$$

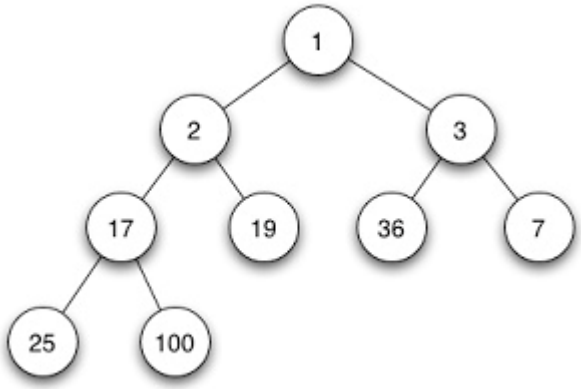
$$GS = TAS + WS * \cos(2 * \pi - (\alpha_w - \alpha_c)) \text{ , eğer } 2 * \pi - (\alpha_w - \alpha_c) < 90°$$

şeklinde hesaplanır. Yere göre hız acil durum türüne göre tutunma hızının üstünde en az ya da yapısalın izin verdiği en üst limitte olabilmektedir. O irtifada, o sıcaklıkta ve o ağırlıktaki en küçük tutunma hızı, değişenlerin gerçek hava hızına etkileri, performans polinomu aracılığı ile hesaplanır.(Stengel, 1993)

Uçaktan uçağa , motordan motora değişiklik gösteren özelliklerden bir tanesi de yakıt tüketimidir. Motor, türüne göre uçak farklı irtifa, ağırlık ve hava sıcaklığında farklı yakıt tüketebilir. Uçağın bu özellikleri her bir değişken için(diğerlerini sabit tutup) ayrı ayrı önceden belirlenmiş sayıda değerlerle ölçülür. Bu değerler, değer sayısı-1 kadar dereceden polinomun katsayılarını belirlemek için kullanılırlar. Oluşan denklemler Gauss-Eliminasyon yöntemi yardımı ile çözülür ve o uçağın yakıt tüketiminin irtifadan, hava sıcaklığından ve ağırlıktan hangi ölçüde etkilendiği belirlenmiş olur. Sezgisel fonksiyonla hesaplanan uzaklık için en yüksek hızda gidildiğinde geçen sürenin yakıt akışıyla çarpılmasından, o an için gereken en az miktarda yakıt hesaplanır ve kısıt olarak kullanılır.

2.4 Optimalleştirme

A* algoritması gereği F değerine göre küçükten büyüğe sıralı komşu gridlerin olması büyük avantaj sağlamaktadır. Öncelikli kuyruk metodu bu avantajı elde etmek için diealdir ve bu metodu ikili en küçük yığını(Binary Min Heap) veri yapısı en az karmaşıklıkla oluşturur. İkili en küçük yığını(İEKY) aslında bir ağaç yapısıdır ve bu ağaç yapısını oluşturmanın iki yolu vardır. Bunlardan ilki bağlı liste ile diğeri dizi kullanarak. İki yönteminde kendine özgü avantajları vardır; bağlı liste yöntemi sınırlı belleğe sahip sistemler için idealken dizi içinde tutmak işlem süresinin en aza indirgenmesi için idealdir. Bu çalışmada bellek kısıtı olmadığı varsayılmıştır. (Cormen, 2000)



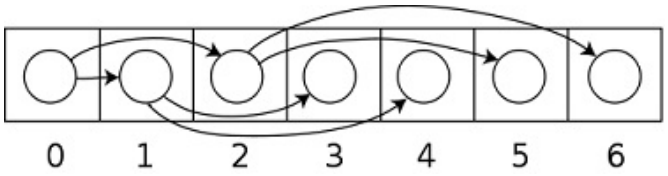
Şekil 2.4.1 İkili En Küçük Yığını

İEKY‘ e bir düğüm ekleme, en düşük F değerine sahip düğümü çıkarma, boş mu dolu mu kontrolü, en düşük F değerine sahip düğümü bulma algoritmaları aşağıdaki gibidir.

İEKY’ nin oluşturulacağı indisi 0’ dan başlayan bir A dizisi olsun;

Bir *i* düğümü sol çocuk düğümüne A[2*i*+1] den sağ çocuk düğümüne A[2*i*+2]

Bir *i* düğümü de kendi ebeveyn düğümüne A[$\left\lfloor \frac{(i-1)}{2} \right\rfloor$] indislerinden erişebilir.



Şekil 2.4.2 İEKY' nin Dizideki Görünüşü

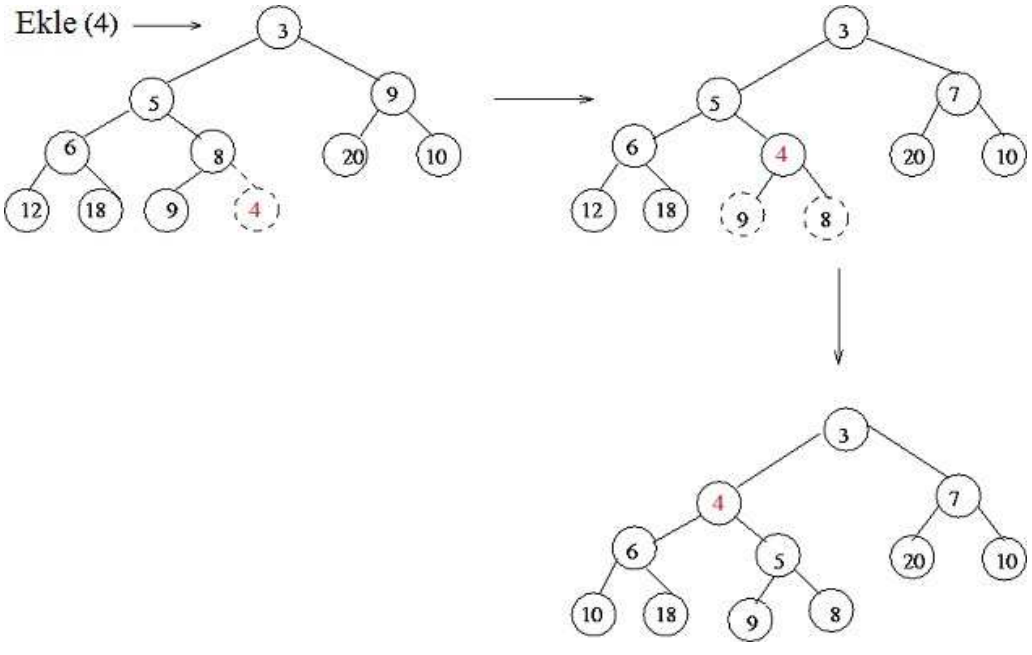
Düğüm Ekleme:

1. Eklenecek düğümü İEKY'nin en alt seviyesine yani dizinin sonuna ekle.

2. Eklenen düğümün F değeriyle ebevyninin F değeri karşılaştır;

   a. Eğer yeni eklenen düğümün F değeri daha büyükse dur.

   b. Değilse, yeni eklenen düğümle ebeveyn düğümünü yer değiştir ve 2. adıma geri dön.



Şekil 2.4.3 Düğüm Ekleme

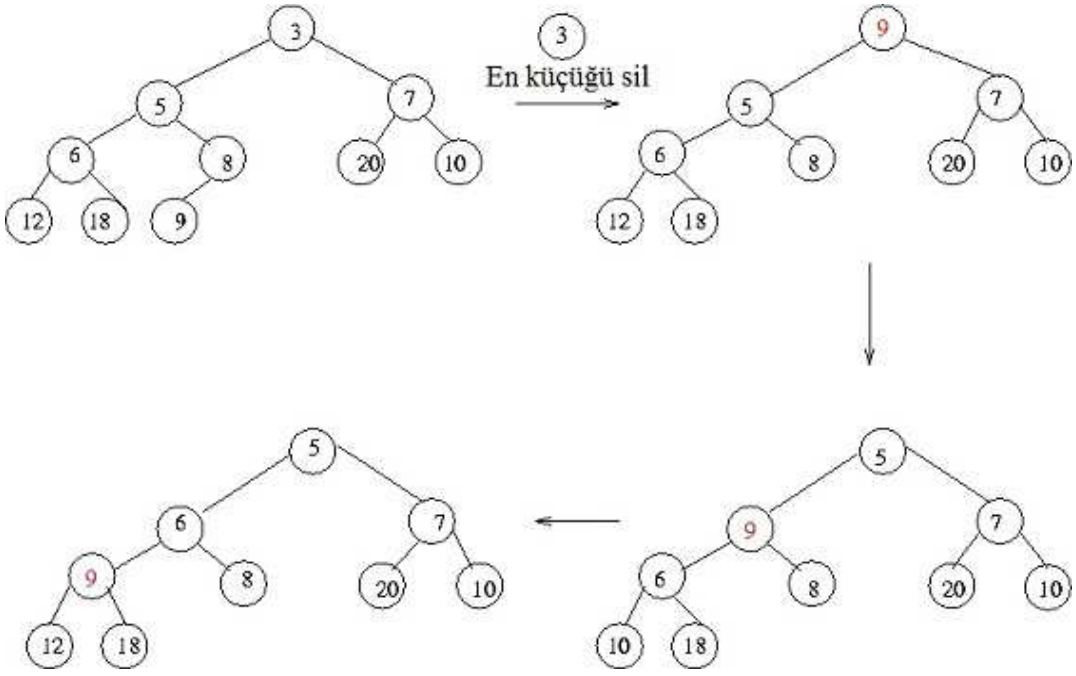
Düğüm Çıkarma:

1. Dizideki son düğümü kök düğüm yap.

2. Yeni kök düğümün F değeri ile çocuk düğümlerin F değerlerini karşılaştır;

    a) Eğer kök düğümün F değeri çocuk düğümlerin F değerinden küçükse dur.

    b) Değilse, çocuk düğümle kök düğümü yer değiştir ve 2. adıma dön.



Şekil 2.4.4 Düğüm Çıkarma

Boş mu dolu mu Kontrolü:

1. Dizinin ilk indisi yani 0. indiste bir düğüm varsa dolu yoksa boş döndür.

En küçük f değerine sahip düğümü Bulma:

1. Kök düğümün indisini döndür.

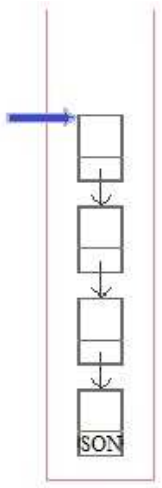
Algoritmalardan da görüldüğü gibi yeni düğüm ekleme, çıkarma ve en az f değerine sahip düğümü alma, boş olup olmama kontrolü karmaşıklıkları Çizelge 2.4.1' deki gibidir.

Çizelge 2.4.1 İEKY yapısının karmaşıklık tablosu

İşlem	Karmaşıklık
Boşmu()	**O(1)**
enKüçük()	**O(1)**
Ekle()	**O(log n)**
enKüçükÇıkar()	**O(log n)**

Görüldüğü gibi İEKY tüm düğümleri sıralamadığı için ki bu durum en küçük düğümü bulmak için yeterlidir, hızlı sıralama(Quicksort), kümeli sıralama(Mergesort) ve yığın sıralama(Heapsort) gibi karmaşıklığı O(n log n) olan en hızlı sıralama algoritmalarından daha hızlı çalışmaktadır.

A* algoritması bittikten sonra yani hedef gride ulaşıldıktan sonra geriye, başlangıç gridine doğru gidilerek rotanın oluşturulması, fazladan gidilen gridlerin elenmesi gerekmektedir. Bu işlem için yığın veri yapısı, bağlı liste kullanılarak oluşturulmuştur. Bunun amacı bağlı liste şeklinde tutulan yığının, dizide tutulan yığına göre geri izlenebilirlik açısından daha az hesap gerektirmesidir.



Şekil 2.4.5 Yığın veri yapısının bağlı liste gösterimi

3. DENEYSEL BULGULAR

3.1 Deney Koşulları

Yapılan bu çalışmanın uygulanabilir ve kullanılabilir olması için deneyde olabildiğince gerçekçi veriler kullanılmaya çalışılmıştır. Deneyde bellek sıkıntısı olmadığı varsayılmış ve program, çalışma süresine göre optimize edilmiştir.

Okyanus aşırı uçuşlarda kıtadan ayrılırken belirlenmiş hava yollarına uyma kısıtı kalktığı ve bu da arama uzayını genişlettiği için okyanus üzerinde bir acil durum yaşanmış gibi programa veri sağlanacaktır. En kötü durum senaryosu için de en büyük okyanus olan pasifik okyanusu değerleri kullanılacaktır.

89.812.800NM^2 (10692 x 8400)lik bir uçuş alanından(Pasifik Okyanusunun alanı 63.800.000NM^2' dir.) 12NM' lık hassaslıkla oluşturulan 700 x 892 (m x n)' lik bir grid yapısı Şekil 3.1.1' deki gibidir.

623507	**623508**	**…**					**624399**
…							
…							
…							
892	**893**						
0	**1**	**2**	**3**	**4**	**5**	**…**	**891**

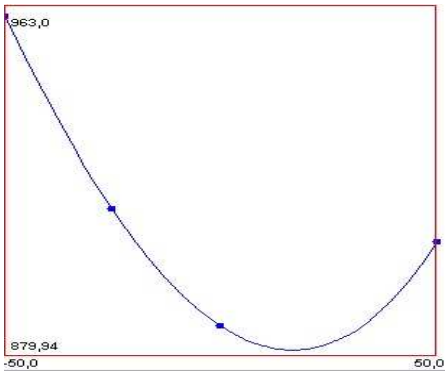
Şekil 3.1.1 Uçuş Alanı

Hassaslığın 12 NM seçilmesinin sebebi, ortalama 200 knots ile giden bir uçağın bu mesafeyi 3,6 dakikada alabilmesi, enlem ve boylam olarak da 0,2 derecelik olup, karar verme süresi bakımından kabul edilebilir olmasıdır. Böylelikle 12 NM hassaslığında gelen hava durumu tahminleriyle de uyum sağlamaktadır. En yüksek irtifa 10NM alınmış ve 1NM' lık 10 eşit parçaya bölünmüştür. Hava tahminleri farklı yükselikler ve 3 saatlik zaman aralıkları için sağlanmıştır. Gridler arası mesafenin hesaplanması için kullanılan büyük çember uzaklığı, iki gridin merkez noktalarını kullanır. Bu hesaplama için gerekli olan Dünya' nın yarıçapı 3443.8984NM alınmıştır. (McNamara, 2004; Moir 2006; Moir 2008)

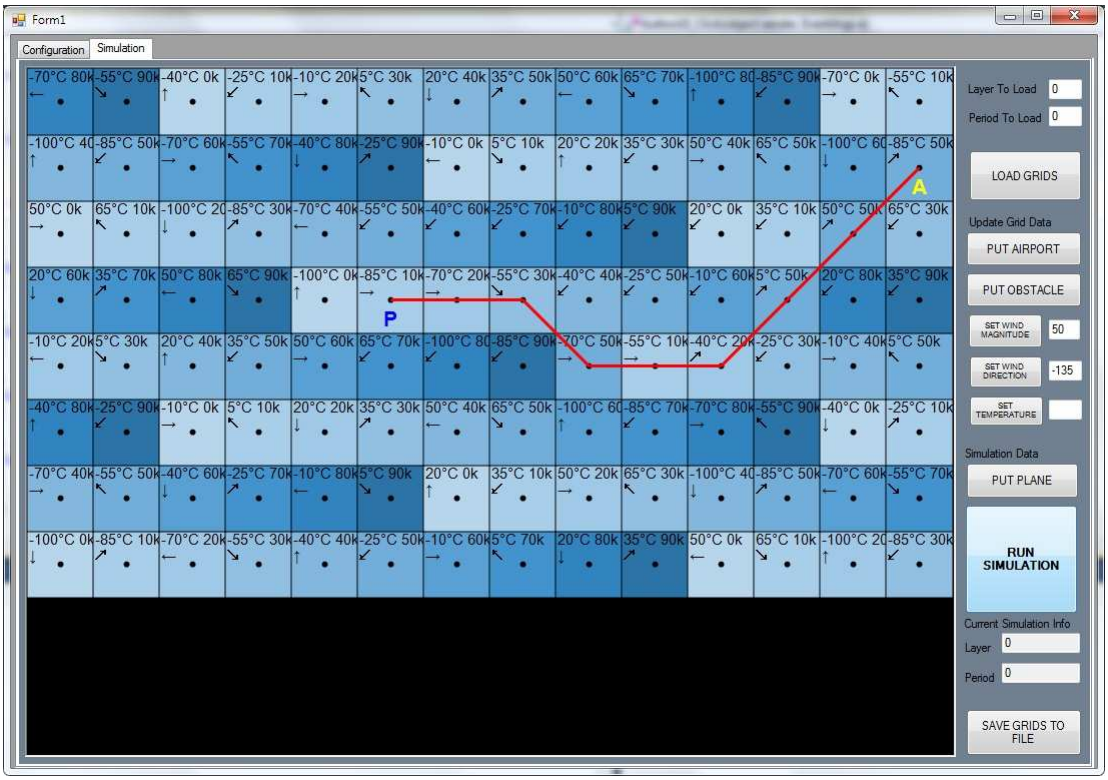
Yakıt akışı ve calibre edilmiş hava hızı performans polinomları, gerçek bir uçağın performans polinomlarının katsayıları uygun şekilde değiştirilerek oluşturulmuştur(Bkz. Çizelge 3.1.1). İniş için gerekli en az yakıt 4000 galon, iniş için gerekli en az mesafe 4NM alınmıştır. Yapısalı etkilemeyecek en yüksek hız 300 knots, kabul edilebilir en yüksek rüzgar hızı da 50 knots alınmıştır. Hava radarı ile hava tahmini arasındaki kayma 1 alınmış, yani örtüştüğü varsayılmıştır.

Uçağın boş ağırlığı 70.000 lb, en fazla yakıt miktarı da 80.000 lb olarak alınmıştır.

Çizelge 3.1.1 Hava Sıcaklığının Yakıt Akışına Etkisi



3.2 Deney Sonuçları



Şekil 3.2.1 Simülatör

Çizelge 3.2.1 Deney Sonuçları

	Yakıt Sızıntısı	En kısa sürede iniş
624400(x10 katman)	30,006 s	29,3965 s
	1350 MB	1350 MB
112(x10 katman)	3 ms	1 ms
	2,5 MB	2,5 MB

4. SONUÇ

Görüldüğü gibi önceki çalışmalar hava koşullarının dezavantajlarından kaçınılarak yapılmıştır, bu çalışmada ise hava koşullarını avantaja dönüştürebilmesine amaçlanmıştır, bu şekilde yakıttan tasarruf sağlanması ve olası bir yakıt sızıntısı acil durumunda kabul edilebilir en yüksek hızla en az yakıt tüketecek şekilde rota planlaması yapılması amaçlanmıştır.

Gerçeğe olabildiğince yakın veriler kullanılmış, çalışma zamanını kısaltmak için program optimize edilmiş, hesaplamalardaki detaylardan kaçınılmamış ve optimum rota, kabul edilebilir bir sürede bulunmuştur.

5. ÖNERİLER

Hava durumu tahminlerinin girdi olarak sağlanabilmesinden ötürü, rotanın planlanması statik olarak yapılmıştır. Yani belirli periyotlarla hava durumu bilgilerinin bilinmesi bir sonraki adımın daha o gride gitmeden göz önüne alınabilmesini sağlamıştır. Eğer yapılacak çalışmada düğümlerin durumları(gidilebilir – gidilemez veya gitme maliyetleri) o düğüme gidildiğinde tahmin edilenden farklı olacağı varsayılıyorsa bu rota planlaması dinamiktir ve anlık verilere göre sürekli yapılmalıdır. Böyle durumlar için geliştirilmiş artımsal A* algoritmalarından D* Lite (Dynamic A* Lite) (Koenig, Likhachev, 2002) algoritması kullanılabilir. Dinamik rota bulma çalışmaları çoğunlukla robotbilim ile uğraşan araştırmacılar tarafından kullanılmaktadır.

EK AÇIKLAMALAR

Bu kısımda yapılan çalışmanın gerçekçi olabilmesi için kullanılan verilerin ve deneysel bulgulardaki değerlerin açıklamaları, bunun dışında ismi verilmiş fakat açıklaması yapılmamış terimlerin ve cihazların açıklamaları yer almaktadır.

Altimetre : Deniz seviyesine göre bir yerin yüksekliğini ölçebilen özel bir barometre.

Hava Veri Bilgisayarı(ADC) : Gerçek hava hızını, irtifayı ve GPS' i kontrol eder.

Coğrafi Kuzey(True North): Kuzey kutbunda tam 90 derece kuzey noktasıdır. Dünyanın küre biçiminde olması dolayısıyla sürekli kuzeye gidilirse bir noktada daha fazla kuzeye gidilemeyecek, dünyanın en tepesi olarak adlandırılabilecek bir nokta bulunması gerekir.

Pito Tüpü: Akışkanın hızını ölçmek için kullanılan basınç sensörü barındıran tüp.

KAYNAKLAR DİZİNİ

Aeronautical Inc, 2006, Advanced Flight Management System, ARINC.

Cary, E. and Spitzer, R., 2001, The Avionics Handbook, CRC Press LLC, Boca Raton, 542p.

Cook, M. B. and Smallman, H. S., 2008, When Plans Change: Task Analysis with Navy UAV Operators, Display Requirements, and UAV Re-Routing Taxonomy, Pacific Science & Engineering Group, California, 29p

Cormen, T. H., Leiserson, C. E., Rivest R. L. and Stein C., 2000, Introduction to Algorithms, McGraw-Hill, Columbus, 1003p.

Ferguson, M., 1999, GPS Land-Navigation, Glassford Publication.

Gill, J. A., 1995, Flight Control Computer Development through Application of Software Safety Technology, Naval Air Warfare Center Aircraft Division.

Hart, P. E., Nilsson, N. J. and Raphael, B., 1968**,** A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics, California, 8p

IA State University, 2010, Geodesics∗, IA State University, IA State, 7p

Koenig, S. and Likhachev, M., 2002, D* Lite, American Association for Artificial Intelligence, Atlanta, 8p

Krozel, J., Lee C. and Mitchell J.S.B., 2006, Turn-Constrained RoutePlanning for AvoidingHazardous Weather, Air Traffic Control Quarterly, New York, 24p

McNamara, J., 2004, GPS for Dummies, Wiley Publishing, Inc.

Meuleau, N., Plaunt, C. and Smith, D. E., 2008, Emergency Landing Planning for Damaged Aircraft, NASA Ames Research Center**,** California, 8p

Moir, I. and Seabridge, A. G., 2006, Military Avionics Systems, John Wiley & Sons Ltd.

Moir, I. and Seabridge, A. G., 2008, Aircraft Systems, John Wiley & Sons Ltd.

Nabiyev V.V., 2005, Yapay Zeka Problemler-Yöntemler-Algoritmalar, Seçkin Yayıncılık.

Nabiyev V.V., 2007, Algoritmalar Teoriden Uygulamalara, Seçkin Yayıncılık.

Russell, S. and Nowig, P., 2003**,** Artificial Intelligence: A Modern Approach, Prentice Hall.

Santoso, L. W., Setiawan, A. and Prajog, A. K., 2010, Performance Analysis of Dijkstra, A* and Ant Algorithm for Finding Optimal Path, Petra Christian University, Surabaya. 8p

Stefanakis, E. and Kavouras, M., 2002, Navigating In Space Under Constraints, International Journal of Pure and Applied Mathematics, Athens, 22p

Stengel, R. F., 1993, Toward Intelligent Flight Control, IEEE Transactions on Systems, Man, and Cybernetics. 23(6).

KAYNAKLAR DİZİNİ (devam)

Airbus, Flight Management Systems on Commercial Aircraft - Past, Present and Future (Airbus), http://www.airbus.com/store/mm_repository/pdf/att00012529/media_object_file_fast_42_p2_p7.pdf (Erişim tarihi: 02 Haziran 2011)

Lester, P., 2005 , A* Pathfinding for Beginners**,** Policy Almanac, http://www.policyalmanac.org/games/aStarTutorial.htm (Erişim Tarihi: 02 Haziran 2011)

UTED, "Flight Management System", http://www.uted.org/dergi/2009/mart/KemalK_ZaferU/fms.htm (Erişim tarihi: 8 Ocak 2011)

Wikipedia, "Flight Plan", http://en.wikipedia.org/wiki/Flight_plan (Erişim tarihi: 8 Ocak 2011)

Wikipedia, "Flight Management System", http://en.wikipedia.org/wiki/Flight_management_system (Erişim tarihi: 8 Ocak 2011)

Wikipedia, "Aircraft flight control system" http://en.wikipedia.org/wiki/Aircraft_flight_control_system (Erişim tarihi: 8 Ocak 2011)

ÖZGEÇMİŞ

BAVUNOĞLU, Harun, Mustafa oğlu, 14/11/1984 tarihinde İzmir' de doğdu. Dokuz Eylül İlköğretim Okulu' nda okudu. 1995 yılında Bornova Anadolu Lisesi Almanca bölümüne başladı ve 2002 yılında bu liseyi bitirdi. Aynı yıl Ege Üniversitesi Matematik Bölümüne başladı.

3. sınıfta Bilgisayar Bilimleri opsiyonunu seçti ve 2007 yılında mezun oldu. Aynı yıl aynı bölümde yüksek lisansına başladı. 2008 – 2009 arasında Almanya' da savunma sanayinde staj yapıp, 2009 yılında TUSAŞ Havacılıkta çalışmaya başladı.

EKLER

EK 1

Kaynak Kod

```
/****************************
 * Ege Üniversitesi         *
 ****************************
 *     Harun Bavunoglu      *
 *                          *
 *     Yüksek Lisans Tezi   *
 *     03.06.2011           *
 *                          *
 *     entryPoint.c         *
 ****************************/
#include "grid.h"
#include "Configuration.h"
#include <stdlib.h>

extern void getNearestArpt(UInt32, GridCell**, Configuration*, UInt32*);
extern UInt32 findShortestPath(GridCell*, GridCell*, GridCell**,
Configuration*);

void __declspec( dllexport ) entryPointEmergency(GridCell* currentGrid,
Configuration* config, GridCell** grids, UInt32* nearestArpts)
{
	// get n(comes from config) nearest airports grids
	// if you have FMS skip this step
	getNearestArpt(currentGrid->gridNumber, grids, config, nearestArpts);

	findShortestPath(currentGrid, &grids[0][nearestArpts[0]], grids,
config);
}

/****************************
 * Ege Üniversitesi         *
 ****************************
 *     Harun Bavunoglu      *
 *                          *
 *     Yüksek Lisans Tezi   *
 *     03.06.2011           *
 *                          *
 *     grids.c              *
 ****************************/
#include <stdlib.h>
#include "grid.h"
#include "Configuration.h"
```

```
// Spiral order traversal algorithm O(n)
void getNearestArpt(UInt32 currentGrid, GridCell** grids, Configuration*
config, UInt32* nearestArpts)
{
	UInt32		diff,i;
	UInt32		tempGrid;
	UInt8		nearestIndex = 0, result = 1;
	UInt16		rowC;


	rowC = config->rowWidth;
	tempGrid = currentGrid;


	for(diff = 1; ; diff += 2)
	{
		// just one step up to start new spiral
		if(nearestIndex == config->nearestLimit)return;
		tempGrid += rowC;
		if(tempGrid > rowC * config->heightRes)continue;
		if(grids[0][tempGrid].hasAirport == 1)
		{
			nearestArpts[nearestIndex] = tempGrid;
			nearestIndex++;
		}

		// expand to the right
		for(i = 1; i <= diff ; i++)
		{
			if(nearestIndex == config->nearestLimit)return;
			tempGrid++;
			if((tempGrid % rowC == 0) || (tempGrid > config-
>flightArea))continue;
			if(grids[0][tempGrid].hasAirport == 1)
			{
				nearestArpts[nearestIndex] = tempGrid;
				nearestIndex++;
			}
		}

		//expand to the down
		for(i = 1; i <= diff + 1 ; i++)
		{
			if(nearestIndex == config->nearestLimit)return;
			tempGrid -= rowC;
			if((tempGrid > config->flightArea)||(tempGrid % rowC ==
0))continue;
			if(grids[0][tempGrid].hasAirport == 1)
			{
				nearestArpts[nearestIndex] = tempGrid;
				nearestIndex++;
			}
		}

		//expand to the left
```

```
                for(i = 1; i <= diff + 1 ; i++)
                {
                        if(nearestIndex == config->nearestLimit)return;
                        tempGrid--;
                        if((tempGrid > config->flightArea)||((tempGrid+1) % rowC
== 0))continue;
                        if(grids[0][tempGrid].hasAirport == 1)
                        {
                                nearestArpts[nearestIndex] = tempGrid;
                                nearestIndex++;
                        }
                }

                // expand to the up
                for(i = 1; i <= diff + 1 ; i++)
                {
                        if(nearestIndex == config->nearestLimit)return;
                        tempGrid += rowC;
                        if((tempGrid > config->flightArea)||((tempGrid+1) % rowC
== 0))continue;
                        if(grids[0][tempGrid].hasAirport == 1)
                        {
                                nearestArpts[nearestIndex] = tempGrid;
                                nearestIndex++;
                        }
                }
        }
}


UInt8 getNeighbours(GridCell* currentGrid, GridCell* startGrid, GridCell**
grids, GridCell* neighbourArray, Configuration* config)
{
        UInt32 rowC;
        UInt32 i, j, l, neighbourIndex = 0;
        Int32  verTrav[3], horTrav[3];

        rowC = config->rowWidth;

        verTrav[0] = -rowC;
        verTrav[1] =  0;
        verTrav[2] =  rowC;

        horTrav[0] = -1;
        horTrav[1] =  0;
        horTrav[2] =  1;


        for(l = currentGrid->layer - 1; l<=currentGrid->layer + 1; l++)
        {
                for(i = 0; i < 3; i++)
                {
                        for(j = 0; j < 3; j++)
                        {
                                if( ((i==1) && (j==1))||
```

```
                                    ((j==2) && ((currentGrid->gridNumber +
horTrav[j]) % rowC == 0))||
                                    ((j==0) && (currentGrid->gridNumber % rowC
== 0))||
                                    (currentGrid->gridNumber + verTrav[i] +
horTrav[j] > config->flightArea)||
                                    ((currentGrid->gridNumber + verTrav[i] +
horTrav[j] == startGrid->gridNumber) && (l == startGrid->layer))||
                                    (l >= config->layerSize))
                                           continue;

                             if(grids[l][currentGrid->gridNumber + verTrav[i] +
horTrav[j]].isWalkable == 1)
                             {
                                    neighbourArray[neighbourIndex] =
grids[l][currentGrid->gridNumber + verTrav[i] + horTrav[j]];

                                    if(neighbourArray[neighbourIndex].parent ==
0x0)

       grids[neighbourArray[neighbourIndex].layer][neighbourArray[neighbourInde
x].gridNumber].parent = &grids[currentGrid->layer][currentGrid->gridNumber];

                                    neighbourIndex++;
                             }
                      }
               }
       }

       return neighbourIndex;
}

/***************************
 * Ege Üniversitesi        *
 ***************************
 *     Harun Bavunoglu     *
 *                         *
 *     Yüksek Lisans Tezi  *
 *     03.06.2011          *
 *                         *
 *     aStar.c             *
 ***************************/
#include "grid.h"
#include "Configuration.h"
#include <stdlib.h>
#include <string.h>

extern UInt8        getNeighbours(GridCell*, GridCell*, GridCell**, GridCell*,
Configuration*);
extern Float32             g(GridCell*, GridCell*, GridCell*, GridCell**,
Configuration*);
extern Float32             h(GridCell*, GridCell*, GridCell**);
extern void                binaryHeapOpenAdd(GridCell);
extern GridCell            binaryHeapOpenGetLowest();
extern void                binaryHeapOpenInit(UInt32);
```

```
extern void                binaryHeapOpenDelete(GridCell);
extern UInt8         binaryHeapOpenIsEmpty();
extern void                pushStack(GridCell*);


void backtraceAndSmoothPath(GridCell* goalGrid)
{
	//use stack to get path ordered
	GridCell* temp;

	temp = goalGrid;

	do
	{
		pushStack(temp);
		temp = temp->parent;
	}while(temp != 0x0);
}


UInt32 findShortestPath(GridCell* currentGrid, GridCell* goalGrid, GridCell**
grids, Configuration* config)
{

	/*******************************************************/
	GridCell	startGrid;
	UInt8		i, neighbourCount=0;
	GridCell*	neighbourArray;

	startGrid = *currentGrid;

	neighbourArray = (GridCell*)calloc(NEIGHBOUR_COUNT, sizeof(GridCell));

	// Do some quick checks
	/***********************/
	if(currentGrid == goalGrid)
		return 0;

	if(goalGrid->isWalkable == 0)
		return -1;

	/*******************************************************/

	binaryHeapOpenInit(config->flightArea * config->layerSize);
	// InitClosedList();

	//1) Add the starting square (or node) to the open list.
	binaryHeapOpenAdd(*currentGrid);

	//2) Repeat the following:
	do
	{
		//a) Look for the lowest F cost square on the open list. We refer
to this as the current square.
		*currentGrid = binaryHeapOpenGetLowest();
```

```
		//b) Switch it to the closed list.
		//arrayClosedAdd(*currentGrid);
		binaryHeapOpenDelete(*currentGrid);

		//c) For each of the 8 squares adjacent to this current square …
		memset(neighbourArray, 0, NEIGHBOUR_COUNT * sizeof(GridCell));
		neighbourCount = getNeighbours(currentGrid, &startGrid, grids,
neighbourArray, config);
		for(i = 0; i < neighbourCount; i++)
		{
			//If it is not walkable or if it is on the closed list,
ignore it. Otherwise do the following.
			//(done by getting neighbours)

			//If it isn't on the open list, add it to the open list.
Make the current square the parent of this square. Record the F, G, and H costs
of the square.
			if(neighbourArray[i].f == 0)
			{

	grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].g =
g(currentGrid, &neighbourArray[i], goalGrid, grids, config);

	grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].h =
h(&neighbourArray[i], goalGrid, grids);

	grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].f = G_COEFF
* grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].g +

                                                                H_COEFF *
grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].h;

	binaryHeapOpenAdd(grids[neighbourArray[i].layer][neighbourArray[i].gridN
umber]);
			}
			/*If it is on the open list already, check to see if this
path to that square is better, using G cost as the measure.
			A lower G cost means that this is a better path. If so,
change the parent of the square to the current square, and recalculate the G
and F scores of the square.
			If you are keeping your open list sorted by F score, you
may need to resort the list to account for the change.*/
			else
			{
				if(grids[currentGrid->layer][currentGrid-
>gridNumber].g <
grids[neighbourArray[i].layer][neighbourArray[i].gridNumber].g)
				{
					*currentGrid = neighbourArray[i];
				}
			}
		 }
	}
	//d) Stop when you:
```

```
	//	Add the target square to the closed list, in which case the path
has been found (see note below), or
	//	Fail to find the target square, and the open list is empty. In
this case, there is no path.
	while(!((currentGrid->layer == goalGrid->layer)&&(currentGrid-
>gridNumber == goalGrid->gridNumber))&&(!binaryHeapOpenIsEmpty()));

	backtraceAndSmoothPath(goalGrid);

	/*********************************************************/
}

/***************************
 * Ege Üniversitesi        *
 ***************************
 *	Harun Bavunoglu		*
 *				*
 *	Yüksek Lisans Tezi	*
 *	03.06.2011		*
 *				*
 *	costCalcs.c		*
 ***************************/
#include "Configuration.h"
#include "grid.h"
#include <math.h>


const Float64 rEarth = 3443.8984;
const Float64 PI = 3.1415;

Configuration*	config;
Float32			totalPassedTime;
Float32			tempDevTAS, tempDevFF;
Float32			weightDevTAS, weightDevFF;
Float32			altDevTAS, altDevFF;


Float32 distanceFlatPlane(Point* source, Point* dest)
{
	Float32		deltaXSquare, deltaYSquare;
	UInt32	deltaZSquare;

	deltaXSquare = (dest->lat - source->lat) * (dest->lat - source->lat);
	deltaYSquare = (dest->lon - source->lon) * (dest->lon - source->lon);
	deltaZSquare = (dest->alt - source->alt) * (dest->alt - source->alt);

	return sqrt(deltaXSquare + deltaYSquare + deltaZSquare);
}

Float32	toRadian(Float32 degree)
{
	return degree * (PI / 180);
}

Float32 toDegree(Float32 radian)
```

```
{
	return radian * (180 / PI);
}

Float32 distanceGreatCircle(Point* source, Point* dest)
{
	Float32 temp;

	temp = acos(sin(source->lat) * sin(dest->lat) + cos(source->lat) *
cos(dest->lat) * cos(dest->lon - source->lon));
	return rEarth * toRadian(temp);
}


Float64 calculateTASTempDev(Int8 oat)
{
	return config->tempDevPolynomTAS[0] * pow((Float32)oat, 4) +
			config->tempDevPolynomTAS[1] * pow((Float32)oat, 3) +
			config->tempDevPolynomTAS[2] * pow((Float32)oat, 2) +
			config->tempDevPolynomTAS[3] * oat +
			config->tempDevPolynomTAS[4];
}

Float64 calculateTASWeightDev(UInt32 gW)
{
	return config->weightDevPolynomTAS[0] * pow((Float32)gW, 4) +
			config->weightDevPolynomTAS[1] * pow((Float32)gW, 3) +
			config->weightDevPolynomTAS[2] * pow((Float32)gW, 2) +
			config->weightDevPolynomTAS[3] * gW +
			config->weightDevPolynomTAS[4];
}

Float64 calculateTASAltDev(Int8 alt)
{
	return config->altDevPolynomTAS[0] * pow((Float32)alt, 4) +
			config->altDevPolynomTAS[1] * pow((Float32)alt, 3) +
			config->altDevPolynomTAS[2] * pow((Float32)alt, 2) +
			config->altDevPolynomTAS[3] * alt +
			config->altDevPolynomTAS[4];
}

Float64 calculateFFTempDev(Int8 oat)
{
	return config->tempDevPolynomFF[0] * pow((Float32)oat, 4) +
			config->tempDevPolynomFF[1] * pow((Float32)oat, 3) +
			config->tempDevPolynomFF[2] * pow((Float32)oat, 2) +
			config->tempDevPolynomFF[3] * oat +
			config->tempDevPolynomFF[4];
}

Float64 calculateFFWeightDev(UInt32 gW)
{
	return config->weightDevPolynomFF[0] * pow((Float32)gW, 4) +
			config->weightDevPolynomFF[1] * pow((Float32)gW, 3) +
			config->weightDevPolynomFF[2] * pow((Float32)gW, 2) +
```

```
                    config->weightDevPolynomFF[3] * gW +
                    config->weightDevPolynomFF[4];
}

Float64 calculateFFAltDev(Int8 alt)
{
	return config->altDevPolynomFF[0] * pow((Float32)alt, 4) +
                    config->altDevPolynomFF[1] * pow((Float32)alt, 3) +
                    config->altDevPolynomFF[2] * pow((Float32)alt, 2) +
                    config->altDevPolynomFF[3] * alt +
                    config->altDevPolynomFF[4];
}

Int16 getCourse(GridCell* source, GridCell* dest)
{
	if(source->gridNumber == dest->gridNumber - config->rowWidth)
		return 0;
	else if(source->gridNumber == dest->gridNumber - config->rowWidth - 1)
		return 45;
	else if(source->gridNumber == dest->gridNumber - 1)
		return 90;
	else if(source->gridNumber == dest->gridNumber + config->rowWidth - 1)
		return 135;
	else if(source->gridNumber == dest->gridNumber + config->rowWidth)
		return 180;
	else if(source->gridNumber == dest->gridNumber + config->rowWidth + 1)
		return 225;
	else if(source->gridNumber == dest->gridNumber + 1)
		return 270;
	else if(source->gridNumber == dest->gridNumber - config->rowWidth + 1)
		return 315;
}

UInt8 getPeriod(GridCell* current, GridCell* dest)
{
#ifdef _DEBUG
	return 0;
#endif
	return (UInt8)((config->currentTime + totalPassedTime) / (WEATHER_PERIOD
* 60));
}

UInt16 getTAS(GridCell* current, Int8 oat)
{
	tempDevTAS		= calculateTASTempDev(oat) / 10.0;
	weightDevTAS  = calculateTASWeightDev(current->fuelOnBoard + config-
>weightWithoutFuel) / 1e15;
	altDevTAS		= calculateTASAltDev(current->centerPoint.alt) /
10.0;

	return current->calibratedAirSpeed + (UInt16)((tempDevTAS + weightDevTAS
+ altDevTAS)/3);
}

UInt16 calculateGS(GridCell* current, GridCell* dest)
```

```
{
	UInt16 tas1, tas2;
	UInt8  ws1, ws2;
	Int16  wd1, wd2;
	Float32 gs1, gs2, effWs1, effWs2;

	if( (dest->weather[getPeriod(current, dest)].temp > config-
>tempUpperLimit)||
		(dest->weather[getPeriod(current, dest)].temp < config-
>tempLowerLimit)||
		(dest->weather[getPeriod(current, dest)].windSpeed > config-
>maxAccSpeed))
			return VERY_BIG;

	tas1 = getTAS(current, current->weather[getPeriod(current, dest)].temp);
	ws1  = current->weather[getPeriod(current, dest)].windSpeed * config-
>driftCoeff;
	wd1  = current->weather[getPeriod(current, dest)].windDir * config-
>driftCoeff;
	effWs1 = ws1 * cos(toRadian(wd1 - getCourse(current, dest)));
	gs1 = tas1 + effWs1;

	tas2 = getTAS(dest, dest->weather[getPeriod(current, dest)].temp);
	tas2+= tas1;
	ws2  = dest->weather[getPeriod(current, dest)].windSpeed * config-
>driftCoeff;
	wd2  = dest->weather[getPeriod(current, dest)].windDir * config-
>driftCoeff;
	effWs2 = ws2 * cos(toRadian(wd2 - getCourse(current, dest)));
	gs2    = tas2 + effWs2;

	if(config->optimizationCharacteristic == 1)
	{
		// max GS
		if((gs1 + gs2)/2 < config->maxAccSpeed)
		{
			// speed up CAS
			dest->calibratedAirSpeed =  tas2;
		}
		else if((gs1 + gs2)/2 > config->maxAccSpeed)
		{
			// slow down CAS
			dest->calibratedAirSpeed =  tas2;
		}
	}
	else if(config->optimizationCharacteristic == 2)
	{
		// min FF -> min CAS
		if(		tas1 + tas2 - tempDevTAS - weightDevTAS - altDevTAS
>
				calculateTASWeightDev(current->fuelOnBoard +
config->weightWithoutFuel))
		{
			// slow down CAS
			dest->calibratedAirSpeed =  current->calibratedAirSpeed -
```

```
                                                                        (tas1 +
tas2 - tempDevTAS - weightDevTAS - altDevTAS -

        calculateTASWeightDev(current->fuelOnBoard + config-
>weightWithoutFuel));
                }
                else if(      tas1 + tas2 - tempDevTAS - weightDevTAS - altDevTAS
<
                                        calculateTASWeightDev(current->fuelOnBoard +
config->weightWithoutFuel))
                {
                        // speed up CAS
                        dest->calibratedAirSpeed =  current->calibratedAirSpeed +

        (calculateTASWeightDev(current->fuelOnBoard + config->weightWithoutFuel)
-
                                                                        (tas1 +
tas2 - tempDevTAS - weightDevTAS - altDevTAS));
                }
        }

        dest->currentFF = dest->calibratedAirSpeed * FF_CAS_COEFF;

        return (gs1 + gs2)/2;
}

Float32 calculateFuelFlow(GridCell* current, GridCell* dest)
{
        tempDevFF     = calculateFFTempDev(dest->weather[getPeriod(current,
dest)].temp);
        weightDevFF   = calculateFFWeightDev(current->fuelOnBoard + config-
>weightWithoutFuel);
        altDevFF      = calculateFFAltDev(dest->centerPoint.alt);

        return current->currentFF + tempDevTAS + weightDevTAS + altDevTAS;
}

Float32 calculatePassedTime(GridCell* current, GridCell* dest)
{
        Float32       dist;
        UInt16 gs;


        dist = distanceGreatCircle(&current->centerPoint, &dest->centerPoint);

        if(config->optimizationCharacteristic == 1)
        {
                gs = config->maxAccSpeed;
        }
        else if(config->optimizationCharacteristic == 2)
        {
                gs = calculateTASWeightDev(current->fuelOnBoard + config-
>weightWithoutFuel);
        }
```

```
	return (dist / gs) * 60;
}

UInt16 calculateMinRequiredFuel(GridCell* dest, GridCell* goal)
{
	return calculatePassedTime(dest, goal) * calculateFuelFlow(dest, goal);
}

UInt8 calculateRemainingFuel(GridCell* dest, GridCell* goal)
{
	if(calculateMinRequiredFuel(dest, goal))
		return 1;
	else
		return 0;
}

Float32	g(GridCell* currentGrid, GridCell* destGrid, GridCell* goalGrid,
GridCell** grids, Configuration* conf)
{
	Float32 dist  = 0.0;
	Float32 ff		= 0.0;
	Float32	a		= 0.0;
	Float32 b		= 0.0;
	Float32 time  = 0.0;
	UInt16 gs		= 0;

	config = conf;

	dist = distanceGreatCircle(&currentGrid->centerPoint, &destGrid-
>centerPoint);

	gs = calculateGS(currentGrid, destGrid);

	if(gs == VERY_BIG)
		return VERY_BIG;

	time = dist / gs;

	if( (destGrid == goalGrid) &&
		(calculateRemainingFuel(currentGrid, destGrid) < config-
>minApprFuel))
		return VERY_BIG;

	if(config->optimizationCharacteristic == 1)
	{
		if(calculateRemainingFuel(currentGrid, destGrid))
			return currentGrid->g + time;
		else
			return VERY_BIG;
	}
	else if(config->optimizationCharacteristic == 2)
	{
		if(calculateRemainingFuel(currentGrid, destGrid))
			return currentGrid->g + ff * time;
		else
```

```
                        return VERY_BIG;
        }
        else
        {
                return dist;
        }
}

Float32       h(GridCell* destGrid, GridCell* goalGrid, GridCell** grids)
{
        return distanceGreatCircle(&destGrid->centerPoint, &goalGrid-
>centerPoint);
}

/***************************
 * Ege Üniversitesi        *
 ***************************
 *     Harun Bavunoglu     *
 *                         *
 *     Yüksek Lisans Tezi  *
 *     03.06.2011          *
 *                         *
 *     binHeap.c           *
 ***************************/
#include "grid.h"
#include <stdlib.h>


GridCell*     heap;
UInt32        heapSize, arraySize;


UInt32 getLeftChildIndex(UInt32 nodeIndex)
{
    return 2 * nodeIndex + 1;
}

UInt32 getRightChildIndex(UInt32 nodeIndex)
{
    return 2 * nodeIndex + 2;
}

UInt32 getParentIndex(UInt32 nodeIndex)
{
    return (nodeIndex - 1) / 2;
}

void binaryHeapOpenInit(UInt32 size)
{
        heap = (GridCell*)malloc(size * sizeof(GridCell));
    heapSize = 0;
    arraySize = size;
}

void binaryHeapOpenShiftUp(UInt32 nodeIndex)
```

```
{
	UInt32		parentIndex;
	GridCell	tmp;

	if (nodeIndex != 0)
	{
		parentIndex = getParentIndex(nodeIndex);
		if (heap[parentIndex].f > heap[nodeIndex].f)
		{
				tmp = heap[parentIndex];
				heap[parentIndex] = heap[nodeIndex];
				heap[nodeIndex] = tmp;
				binaryHeapOpenShiftUp(parentIndex);
		}
	}
}

void binaryHeapOpenAdd(GridCell a)
{
	if (heapSize == arraySize)
		return;
	else
	{
		heapSize++;
		heap[heapSize - 1] = a;
		binaryHeapOpenShiftUp(heapSize - 1);
	}
}

void binaryHeapOpenShiftDown(UInt32 nodeIndex)
{
	UInt32 leftChildIndex, rightChildIndex, minIndex;
	GridCell tmp;


	leftChildIndex = getLeftChildIndex(nodeIndex);

	rightChildIndex = getRightChildIndex(nodeIndex);

	if (rightChildIndex >= heapSize)
	{
		if (leftChildIndex >= heapSize)
			return;
		else
			minIndex = leftChildIndex;
	}
	else
	{
		if (heap[leftChildIndex].f < heap[rightChildIndex].f)
			minIndex = leftChildIndex;
		else
			minIndex = rightChildIndex;
	}
	if (heap[nodeIndex].f > heap[minIndex].f)
	{
```

```
                tmp = heap[minIndex];
                heap[minIndex] = heap[nodeIndex];
                heap[nodeIndex] = tmp;
                binaryHeapOpenShiftDown(minIndex);
        }
}

UInt8 binaryHeapOpenIsEmpty()
{
    return (heapSize == 0);
}

void binaryHeapOpenDelete(GridCell* d)
{
        if (binaryHeapOpenIsEmpty())
                return;
        else
        {
                heap[0] = heap[heapSize - 1];
                heapSize--;
                if (heapSize > 0)
                        binaryHeapOpenShiftDown(0);
        }
}

GridCell* binaryHeapOpenGetLowest()
{
        if (binaryHeapOpenIsEmpty())
                return 0x0;
        else
                return &heap[0];
}

void binaryHeapOpenFree()
{
    free(heap);
}

/****************************
 * Ege Üniversitesi         *
 ****************************
 *     Harun Bavunoglu      *
 *                          *
 *     Yüksek Lisans Tezi   *
 *     03.06.2011           *
 *                          *
 *     stack.c              *
 ****************************/
#include "grid.h"
#include <stdlib.h>


typedef struct node
{
        GridCell* cell;
```

```
	struct node *link;
}stackElement;

stackElement* stackTop;

void initStack()
{
	stackTop = 0x0;
}

void pushStack(GridCell* g)
{
	stackElement *x;
	x=(stackElement*)malloc(sizeof(stackElement));
	x->cell = g;
	x->link = stackTop;
	stackTop = x;
}

GridCell* popStack()
{
	GridCell* a;
	if(stackTop==0x0)
	{
		return 0x0;
	}
	else
	{
		a=stackTop->cell;
		free(stackTop);
		stackTop=stackTop->link;
		return (a);
	}
}
```

(Gill, 1995)