ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

AUTOPILOT SYSTEM AND GROUND STATION SOFTWARE FOR UAV's

M.Sc. THESIS

Selman TOSUNOĞLU

Department of Aeronautics and Astronautics

Interdisciplinary Aeronautical and Astronautical Engineering Graduate Program

January 2013

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

AUTOPILOT SYSTEM AND GROUND STATION SOFTWARE FOR UAV's

M.Sc. THESIS

Selman TOSUNOĞLU
(511081119)

Department of Aeronautics and Astronautics

Interdisciplinary Aeronautical and Astronautical Engineering Graduate Program

Thesis Advisor: Prof. Dr. A. Rüstem ASLAN

January 2013

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

İNSANSIZ HAVA ARAÇLARI İÇİN OTO PİLOT SİSTEMİ VE YER İSTASYONU YAZILIMI

YÜKSEK LİSANS TEZİ

Öğrenci Selman TOSUNOĞLU
(511081119)

Uçak ve uzay Bilimleri Fakültesi
Disiplinlerarası Uçak ve Uzay Mühendisliği Yüksek Lisasns Programı

Tez Danışmanı: Prof. Dr. A. Rüstem ASLAN

Ocak 2013

Selman TOSUNOĞLU, a **M.Sc.**student of ITU **Department of Aeronautics and Astronautics / Interdisciplinary Aeronautical and Astronautical Engineering Graduate Program** student ID **511081119**, successfully defended the **thesis** entitled "**AUTOPILOT SYSTEM AND GROUND STATION SOFTWARE FOR UAV's**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. A. Rüstem ASLAN** İstanbul Technical University	.....
Jury Members :	**Asst. Prof. Dr. Güneş Karabulut Kurt** İstanbul Technical University	.....
	Prof. Dr. Çingiz HACIYEV İstanbul Technical Üniversity	.....

Date of Submission : 22 September 2012
Date of Defense : 23 January 2013

This thesis is lovingly dedicated to my wife Esra. She supported me on each step of the work. I also would like to dedicate this thesis to my son Yunus. His presence inspired me all the way throughout this work.,

FOREWORD

I would like to thank to Assist. Prof. Dr. Turgut Berat KARYOT for making me start working on digital systems design and embedded programming, to Prof. A. Rüstem ASLAN for his encouragement and support on this thesis. I also would like to thank to my manager in Altinay Robotics Company Mr. Can BAYAR for his trust and support during my education . I finally would like to thank to my wife Esra for her support all the way during this thesis.

January 2013

Selman TOSUNOĞLU
Astronautical Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

GPS **:** Global Positioning System
IMU **:** Inertial Measurement Unit
I/O **:** Input Output
UAV **:** Unmanned Aerial Vehicle
PWM **:** Pulse Width Modulation

LIST OF TABLES

Page

LIST OF FIGURES

Page

AUTOPILOT SYSTEM AND GROUND STATION SOFTWARE FOR UAV's

SUMMARY

The aim of this work is to design and build an autopilot system hardware and software including ground station software which is capable of performing simple mission tasks like: flight control ,attitude stabilization,airspeed control,altitude hold, heading control, GPS hold, waypoint flight, fixed waypoint flight, telemetry, map display , monitoring system variables, flight parameter change during flight, Fly by PC and Control by PC and Joystick. The autopilot system can be implemented in to both fixed wing and rotorcraft platforms. To do that, a multidisciplinary approach has been used. The first step of autopilot design task is to choose the suitable microcontroller and peripherals, then the software has to be written for the controller. For the telemetry system a binary communication protocol must be created which is able to make fast two way communication between ground station and the autopilot system. At last ground station software has to be written to monitor and change the system variables and act as a user interface with camera and flight controls for real UAV experience.The most important part of this system is the flexibility to implement it in to various flying platforms. To be able to do that system variables should be monitored and changed during flight. The common peripherals of the system for all the flying platforms are: main controller, GPS system, inertial measurement unit (IMU), 3 axis gyro, 3 axis accelerometer, 3 axis magnetometer, pitot tube sensor for airspeed measurement, pressure sensor for barometric altitude, servo controller, telemetry modem, current and voltage sensors, primary flight control radio receiverAs a result a working autopilot system has been created. UAV market has grown explosively over the past decade. To simplify the Test flight and controller design phase such a system is very useful and less costly than imported competitors. My contribution to this project is to design and build a working autopilot system which is able to perform basic tasks mentioned above, using advanced embedded programming, digital system design and human machine interface design skills.

İNSANSIZ HAVA ARAÇLARI İÇİN OTO PİLOT SİSTEMİ VE YER İSTASYONU YAZILIMI

ÖZET

Bu çalışmanın amacı temel uçuş görevlerini yerine getirebilecek bir oto pilot sisteminin ve yer istasyonu yazılımının gerçekleştirilmesidir. Bu görevler: Uçuş kontrolü, Konum sabitlenmesi, Hava hızı denetimi, Yükseklik sabitlenme denetimi, yönelme denetimi, GPS destekli konum denetimi, Hedef konum uçuşu, Belirlenmiş konumlara uçuş, Kameralı gözetleme, Beşik sistemi denetimi, Uzaktan kamera denetimi, Telemetri, Harita üzerinde gösterim, Sistem değişkenlerinin gösterimi, Uçuş parametrelerinin değiştirilebilmesi, Kişisel bilgisayar ile uçuş, Kişisel bilgisayar ve joystick ile uçuş denetimi dir.

Ototpilot sistemi sabit veya döner kanatlı hava araçlarında kullanılabilecek şekilde geliştirilmiştir. Bunu başarabilmek için yazılım, elektronk mühendisliği, uçak mühendisliği ve kontrol mühendisliği gibi birçok mühendislik disiplininden istifade edilmiştir.

Bu sisteminin gerçekleştirilmesi ile ilgili ilk adım denetleyici ve diğer donanımsal sistem bileşenlerinin seçimidir. Bu seçim projenin kavramsal tasarım aşamasında yapılmaktadır. Seçim sırasında öncelikli kriterler kolay kullanım, yaygın kod desteği , yardım kaynaklarının bulunabilir olması ve denetleyicinin teknik özellikleri gibi kriterlere göre yapılmıştır. Seçilen mikrodenetleyicinin, farklı sensörtipleri ile iletişim kurabilecek yapıda olması öncelikli hedef olmuştur. Sözgelimi telemetri modem, IMU ve GPS, UART haberleşmesi ile otopilota veri göndermekte, basınç sensörü I2C haberleşmeyi kullanmakta, batarya ölçüm sensörleri ise analog olarak veri göndermektedir. Otopilot ana denetleyicisi bütün bu haberleşme yeteneklerine sahip olmalıdır. Diğer önemli bir teknik özellik ise, sayısal giriş çıkış pinlerinde kesmeler oluşturulabilmesi yeteneğidir. Bu kesmeler sayesinde konvansiyonel bir RC alıcının pinlerinden çıkan PWM sinyallerinin okunması mümkün olabilmektedir. Ayrıca mikrodenetleyici sistemin 5V ile çalışıyor olması tercih sebebidir. Bu sayede, bütün araç içerisinde sadece 2 farklı voltaj seviyesi ile çalışmak mümkün olmaktadır. Geleneksel RC bileşenleri 5V seviyesinde çalışmakta ve çoğu zaman motor sürücüleri 5V çevirici içermektedir. Dolayısı ile 5V ile çalışan kontrolcü seçimi, fazladan bir çevirici gereksimini ortadan kaldırmaktadır.

Daha sonraki adım denetleyici kodunun yazılmasıdır. Denetleyici kodunun geliştirilmesindeki ilk adım, sensörler, aktüatörler ve telemetri sisteminin çalışmasına imkan tanıyacak yazılımın geliştirilmesidir. Buyazılımlar c++ kullanılarak geliştirilmiş ve kontrolcüye hex dosyası olarak yüklenmiştir. Geliştirme sırasında adım adım ilerlenerek ara programcıklar oluşturularak geliştirilen sensör kitaplıkların çalışır vaziyette olduklarına dair kontrol noktaları olarak kaydedilmiştir. Bu sayede denemeler sırasında oluşabilecek arızalarda veya beklenmeyen davranışlarda sorunun donanımsal mı yoksa yazılımsal mı olduğu, deneme kodları tarafından anlaşılabilmektedir. Ana denetleyici kodu sürekli bir döngü içerisinde çalışan bir kontrol kodudur. Döngü zaman kritik olan görevleri yerine getirebilecek şekilde optimize edilmiştir. Sensör datalarının alınması veya telemetri işlemleri ana zaman kritik döngünün dışında, farklı yenileme frekanslarındaki döngülerde yapılmaktadır. Bu sayede sözgelimi 5Hz de çalışan bir GPS sensöründen veriler saniyede 100 defa değil olması gerektiği gibi 5 defa sorgulanmaktadır. Diğer bir yaklaşım ise IMU da

olduğu gibi, yoğun matematiksel hesap gerektiren işlemlerin ayrı bir düşük maliyetli işlemci yapılarak sonuçların ana kontrolcüye gönderilmesidir. IMU bileşeni , içerisindeayrı bir düşük maliyetli işlemci barındırmaktadır. Eksen takımı dönüşümlerini ve IMU'nun euler açılarını bu düşük seviye kontrolcü yapmakta ve sonuçları yine geliştirilmiş bir binary protokolüzrinden ana otopilot işlemcisine yollamaktadır. Bu sayede zaman kritik olan ana döngü böylesine bir işlemle meşkul edilmemiş ve ana kontrol frekansı kararlılığı sağlanmıştır. Aktüatörlere ilişkin kod yazımı da deneme kodları oluşturularak gerçekleştirilmiştir. SadeceRC servo motorların sürülmesine imkan tanıyan kod parçaları oluşturulmuş, RC servolarda denenmiş ve kaydedilmiştir. Ayrıca RC alıcının sinyalleriniokuyan diğer bir değişle , kullanıcının elindeki geleneksel RC vericiden gönderilen pozisyon komutlarını okuyan kod ile servoları istenilen açıya getirmeye yarayan kod paralel geliştirilmiştir. Bu kod otopilotun devrede olmadığı sırada çalışan, otopilotu servo ve RC alıcı arasında bir pasif bileşen gibi kullanılmasını sağlayan koddur. Esasen otopilot kullanıcının komutlarını alan ve uçuş senaryosuna göre bunları servolara ileten bir ara işlemcidir. Bu işlemci otopilot devrede değilken sadece bir geçit vazifesi görür.

Son olarak yer istasyonu yazılımının gerçekleştirilmesi gerekmektedir. Yer istasyonu yazılımı ve oto pilot içerisindeki telemetri yazılımı paralel ilerlemesi gereken süreçlerdir. Otopilota telemetri kabiliyeti kazandırmadan uçuş sırasında data almak mümkün olmamakta ve uçuş denetimine ilişkin algoritmalar doğrulanamamaktadır. Bu sebepten telemetri kodunun yazılması kritiktir. Telemetri kodu daha fazla veri taşıyabilmek için ASCII değil binary olarak tasarlanmıştır. Bu sayede sözgelimi 5 basamaklı bir sayının gönderilmesi 5 byte yerine 2 bayt tutmakta ve rf modemin bant genişliği daha optimum kullanılabilmektedir. Bu da birim zamanda otopilottan daha fazla veri alınması ve otopilota daha fazla veri gönderilmesi anlamına gelmektedir. Yer istasyonu UAV uçuş kullanıcı arabirimi olarak çalışmaktadır. Yer istasyonu yazılımı, Delphi yazılım geliştirme platformu kullanılarak gerçekleştirilmiştir. Delphi, Pascal tabanlı ve görsel tasarım unsurları barındıran çok yaygın ve ileri seviye bir programlama dilidir. Bu sayede kullanıcı arabirimi ve arkasında windows işletim sistemi kaynakları kullanan çok gelişmiş yazılımlar geliştirilebilmektedir. Delphi ile geliştirilen yer istasyonu yazılımı, Web üzerinde verilen java tabanlı bir servis olan Google Maps Api v3 ile de haberleşmekte bu sayede kullanıcıya harita desteği sunulmasını sağlamaktadır. Bu harita deteğinin sağlanması da yine temel java bilgisi gerektirmekte ve delphi tabanlı yazılım ile bu servisin haberleşmesini sağlayacak mekanizmalar geliştirilmesini gerektirmektedir. Proje kapsamında GoogleMaps Api kullanılarak da özel kodlar geliştirilmiş, uçuş rotasının görsel gösterimi, uçağın pozisyonu , yönelmesi ve gidilmesi gereken doğrultunun gösterimi gibi kullanıcı arayüzü gösterimine ilişkin java kodları oluşturulmuştur. Bütun bu oluşturulan kodlar, otopilot un bir uydu görüntüsü üzerinden izlenmesini sağlamakta ve uçuş görevi oluşturulmasını kolaylaştırmaktadır. Bunun yanında yer istasyonu sayesinde otopilota ilişkin parametreler uçuş sırasında değiştirilebilmektedir. Bu parametreler yeni bir görev tanımına ilişkin konum değişiklikleri olabildiği gibi, kontrol sistemine ilişkin kazanç katsayıları da olailmektedir. Ayrıca kamera pozisyonu, uçak eyleyicilerinin aktive edilmesi de bu mekanizma üzerinden sağlanmaktadır. Joystick ile uçuş sırasında, telemetri modem üzerinden otopilota kontrol yüzeylerinin açılarına ilişkin komutlar gönderilmektedir.

Geliştirilen sistemin diğer bir önemli özelliği değişik uçuş platformlarına uygulanabilirliğidir. Sistem bileşenleri buna göre seçilmiştir. Bütün uçan platformlarda kullanılması düşünülen oto pilot bileşenleri şunlardır: Ana Denetleyici,

GPS, Ataletsel Ölçüm Birimi, 3 eksen jiroskop, 3 eksen ivme sensörü, 3 eksen manyetik sensor, Pitot Tüpü Sensörü, Altimetre Basınç Sensörü, Servo Denetleyicisi, Telemetri Modem, Akım ve Gerilim Sensörleri ve Ana Uçuş Kontrol Alıcısı. Bu bileşenler, döner kanatlı hava araçlarında ve sabit kanatlı hava araçlarında kullanılmaktadır.

Yukarıda bahsedilen bileşenler ayrı ayrı farklı amaçlar için yurtdışı ve yurtiçi pazarda uygun fiyatlarda bulanabilmektedir. Bileşenlerin bir sayısal sistem tasarımı yaklaşımı ile bir araya getirilmesi sayesinde, esnek bir sistem ortaya konmuştur. Geliştirilen sistem herhangi bir marka veya model bileşene bağımlı değildir. Geliştirme sürecindeki bütün bilgi birikimi yerli olup, bileşenlerin tamamına yakınının farklı tedarikçilerden ve veya farklı markalardan ikamesi mümkündür. Bu çalışma neticesinde çalışan bir otomotik pilot sistemi, tamamı ile yerli mühendislik bilgi birikimi ile, tedarikçi ve markadan bağımsız bir biçimde ve düşük maliyetli bileşenler ile geliştirilmiştir. Optimize otopilot kod döngüsü, düşük işlem güçlü kontrolcülerin kullanımına imkan vermiştir.

IHA pazarı geçtiğimiz on yılda ciddi bir büyüme kaydetmiştir. Ülkemizdede bu tip çalışmalar hızlanmıştır. IHA lar ülkemiz için stratejik öneme sahip hava araçlarıdır. Bu hava araçlarının ana kontrol bileşenlerinin yerli imkanlarla üretilebiliyor olması stratejik bir önem arzetmektedir. Bu tür araçların geliştirme ve ar-ge süreçleri yüksek maliyetlidir. Bu süreçlerde düşük maliyetli ve esnek otopilot bileşenlerinin kullanımı sektörde geliştirme maliyetlerini düşürecek ve süreçlerin daha hızlı ilerlemesini sağlayacaktır. Sözgelimi elden atılan bir insansız hava aracının malyetinin büyük bir kısmını otopilot oluşturmaktadır. Test uçuşu ve geliştirme sürecinde harcanan iş gücü ve sistem tasarımı maliyetini minimize etmek adına ithal rakiplerinden daha uygun fiyatlı bir denetleyici geliştirme hedefine başarıyla ulaşılmıştır. Bu tezde gömülü sistem yazılımı, sayısal sistem tasarımı ve kullanıcı ara yüzü tasarımı konusunda ileri becerilerin kullanımı şeklinde katkım olmuştur.

1. INTRODUCTION

There is an increasing interest in small unmanned aerial vehicles (UAV's) in the market. Small UAVs have a relatively short wingspan and light weight. They are expendable, easy to be built and operated. Most of them can be operated by one to two people, or even be hand-carried and handlaunched. (Cambone, 2005). Since the UAV'operate mostly on low altitudes, it is very easy for them to crash on survailence area. To avoid this it is essential to build a robust, easy to operate auto-pilot system. It has to be cheap as well to minimize the operating and/or recovery costs in case of crash or part replacement. An autopilot is a mechanical, electrical, or hydraulic system used to guide a vehicle without assistance from a human being. Autopilots are systems to guide the UAVs in flight with no assistance from human operators. Autopilots were firstly developed for missiles and later extended to aircrafts and ships since 1910s (Sullivan, 2006). Autopilot systems become one of the key elements of modern aviation. Also for UAV market, the autopilot systems play a very important role. In this project, an autopilot system and ground station has been designed, integrated in to an aircraft and flown successfully. This project is not proposing an optimized control algorithm but offers a full system which needs further parameterization according to integrated flying platform. There are many choices when it comes to control theory. In this thesis a linear PID controller have been used. This controller is being used since 1890's. The first practical example of PID controller is implemented by Elmer Sperry (Bennett,1984).Even it is a linear controller it has been used in nonlinear systems such as fixed wing UAV' as well. (Albaker and Rahim,2011). In the UAV development phase, it is essential to monitor all the system variables during flight. The system variables can be flight stabilization control parameters, the variables from the sensors like pitch roll angles, altitude, or GPS coordinates of the aircraft. The telemetry systems are therefore very important part of an autopilot system during UAV development. A real-time telemetry system and autopilot system and ground control station is within the scope of this thesis. .

2. SYSTEM DESCRIPTION

This thesis is aiming to propose a working system for UAV developers. This system can be implemented in to mass produced platform as well. The proposed system consists of Hardware and Software components. For the proposed system, several design phases has to be fallowed. These design phases can be summarized as: Controller and Peripheral Selection, PCB Design and Prototyping, Controller Software Design, Ground Station Software Design, Ground Station Hardware Design, Ground Station Software Design, Prototyping and Test flight, Whole system can be summarized as follows: Autopilot, Ground Station, Test Platform.

2.1 Autopilot

Figure 2.1 describes the autopilot sub components.

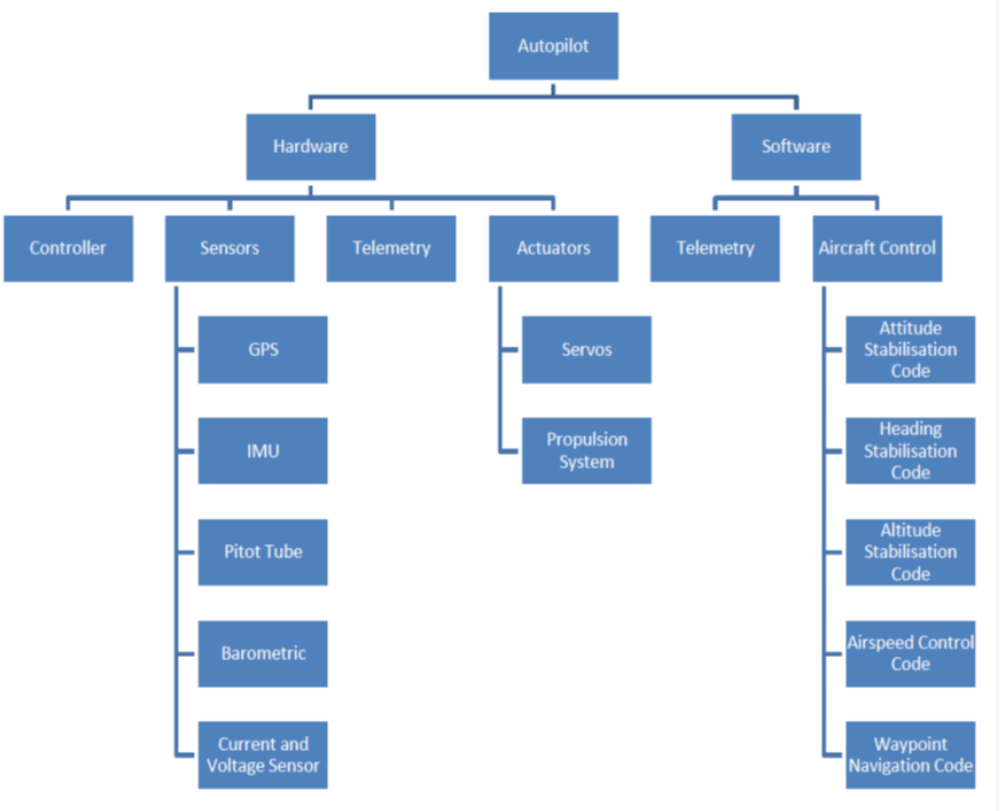

Figure 2.1 : Autopilot SubComponent's Tree.

2.1.1 Autopilot Hardware

Autopilot hardware for the UAV's can be divided in to 4 sections. These are controller, sensors, telemetry unit and actuators.

2.1.1.1 Autopilot Controller

Controller is a computer with communication modules, digital and analog I/O's. It is responsible of almost every action of an autopilot. It is a master CPU that is running in a loop to get signals from sensors or communication ports and sends commands to servo actuators or motor controller. The selection criteria for this project have been the availability in the market, Easy programmability and library availability for peripherals, Processing speed, I/O compatibility, Communication system compatibility and Cost.

As controller, the ATmega2560 chipset is selected. This chipset is used on the popular development platform Arduino since years. It is very easy to find in the market with wide variety of programming support and code samples. The processing speed is 16 MHz, which is enough to perform a real-time control task. There are 4 UARTS and 1 I2C and 2 SPI serial communication ports. The entire I/O and communication task can be done without having big knowledge on microcontroller registers. Figure 2.2 shows the controller board. The easy to code native libraries allows programmer to use simple command for communication and I/O tasks. Autopilot controller its self is not ready to be used in an aircraft, a new PCB design has to made, which allows sensors and actuators connect to the master controller. PCB design has been made by a freeware software called "Desigspark PCB v3.0". The goal of this PCB design is to house the master controller card, which has been purchased separately, and to contain the connectors. Figure 2.3 shows the designed PCB board for master controller.

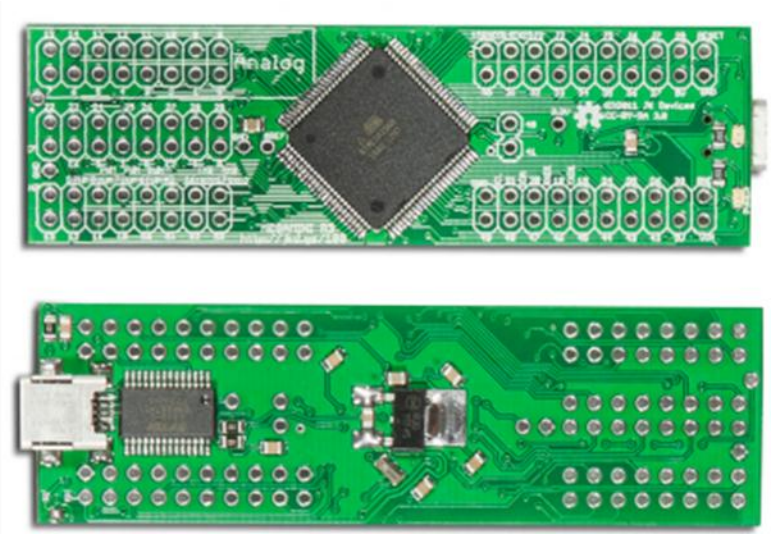

Figure 2.2 : Atmega Controller

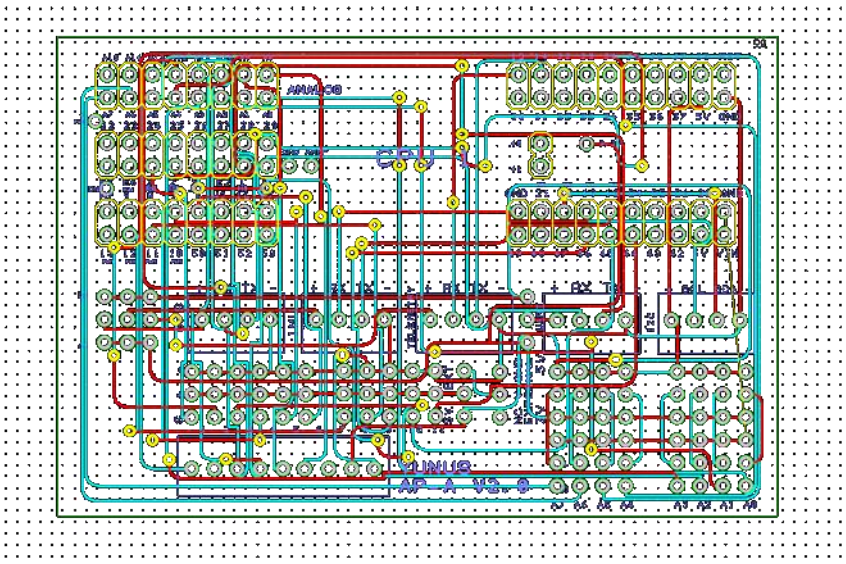

Figure 2.3 : Autopilot PCB Design

The designed PCB board has been sent to manufacturer for production. Figure 2.4 shows the PCB without welded components on, just after the manufacturing process.

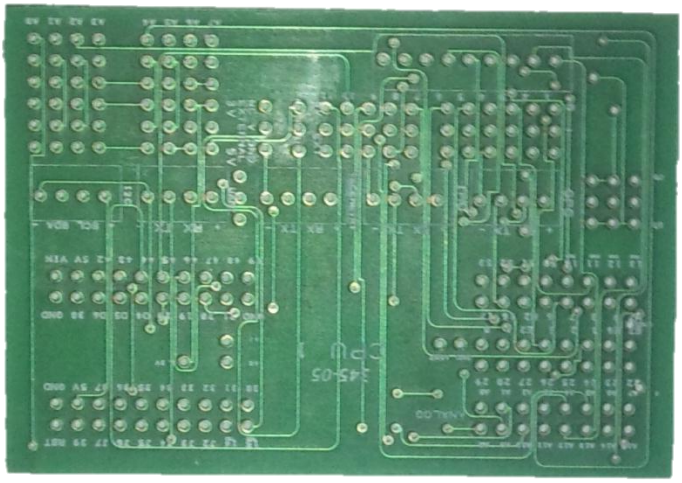

Figure 2.4 : Autopilot PCB after Manufacturing.

After manufacturing process, the connectors and basic components have been soldered on the PCB. Figure 2.5 show the autopilot hardware with connectors and master controller attached.

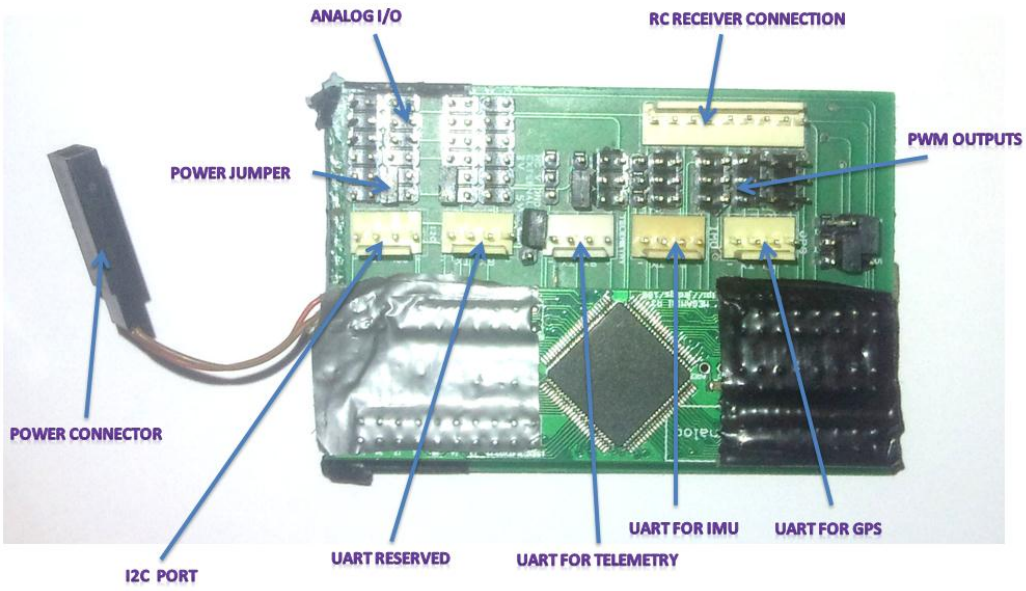

Figure 2.5 : Autopilot PCB after soldering components

2.1.1.2 Autopilot Sensors

As listed in the figure 2.1 there are several sensor needed for autopilot system.

GPS

GPS is a global positioning system. It is calculating the position of the system on earth by getting signals from various satellites and giving this information as latitude, longitude, altitude and heading through serial communication to master controller. Figure 2.6 shows the module used in the autopilot system. The communication between GPS and the controller is done via serial communication at 57.600 bps. The serial port #2 is used for communication. The features are listed as follows. MediaTek MT3329 solution, 5Hz output, 57600bps TTL serial interface

Figure 2.6 : GPS Module

GPS starts sending packet after power up in a special ASCII format. The GPS NMEA sentences needs to be parsed via host controller to get information like, Latitude, Longitude, Altitude, Heading, Satellites availability etc. Below Figure 2.7 is an example of NMEA sentences coming from GPS. The details of GPS NMEA protocol are not within the scope of this thesis.

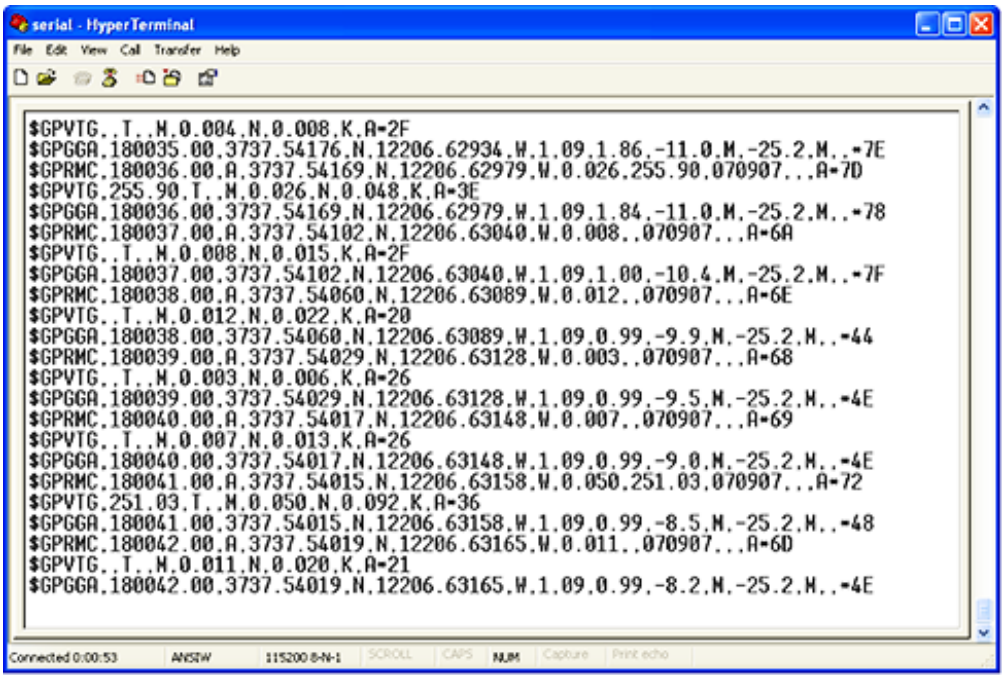

Figure 2.7 : NMEA Sentences

Auto Pilot Inertial Measurement Unit (IMU)

Inertial Measurement Unit is one of the key components of an autopilot system. The IMU is calculating the Euler angles and sending this data trough serial communication to the master controller unit. A processor inside the IMU system performs the calculation of angles and rates without bringing extra calculation effort to the main controller. Programmable IMU has been selected as the attitude sensor. System has been reprogrammed to allow binary communication between controllers with an update rate of 50Hz. The communication rate is 38.400bps. Below figure 2.8 shows the IMU system used in this project.

Figure 2.8 : IMU

IMU systems are calculating airplanes euler angels and angular rates.

Airspeed Sensor

Airspeed sensor consists of a pitot tube and differential pressure sensor. It produces an analog signal and read throug master controllers ADC. In autopilot systems for fixed wing aircraft systems the Lift force strictly depends on airspeed. Therefore the controller needs to know the airspeed to generate the proper moment and aileron or elevator deflection for the given airspeed. Below figure 2.9 shows the airspeed sensor and the pitot tubeused in this thesis.

Figure 2.9 : Airspeed Sensor

Barometric Pressure Sensor

The altitude value from GPS is a low frequency unreliable data. The accuracy of GPS altitude information depends on geographical and atmospheric conditions and the accuracy is between 20 to 100 meters in most cases. Another disadvantage of this system is its latency. The generated altitude signal is delaying up to 2 seconds. To correct this we need to calculate the altitude from a pressure sensor. By using suitable atmosphere models, we can calculate the altitude with an accuracy of 30cm. For altitude hold action, we need a responsive sensor like barometric pressure sensor to generate a reacting movement by autopilot. Below figure 2.10 shows the barometric pressure sensor BMP085. The desired communication protocol with this sensor is I'C.

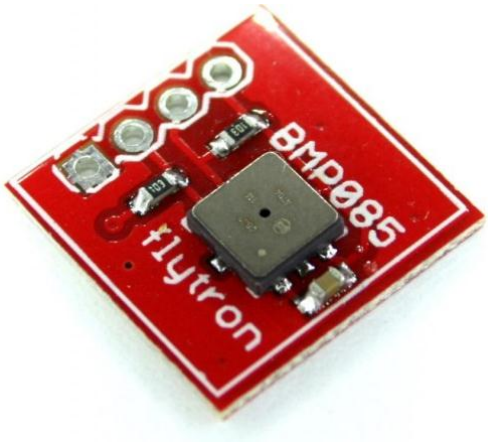

Figure 2.10 : Barometric Pressure Sensor

Autopilot Voltage and Current Sensor

For an UAV system most of the devices are relying on battery energy. In this case, it is a vital job for the controller to calculate the energy remaining in the batteries. We cannot just add more battery to be safe thus; the batteries are heavy which is costly to fly. To be aware of remaining energy in the batteries we need to get the current and voltage information from the battery. By using a method called "cloumb counting", we can calculate the Ampere-hours by measuring the current flow and integrating it by time. In most cases the energy is proportional to the current flow per time unit. The sensor number can be as many as the batteries installed. In modern UAV systems the batteries may also be the main source of energy even for the Propulsion system. This increases the importance of the current and voltage measurement in such an autopilot system. Below figure 2.11 show the current and voltage sensor.

This sensor has 2 analog outputs. The current sensor output gives 5V for 50A and 0V for 0A. The other output is directly connected to the battery + terminal and connected to autopilot via a voltage divider.

Figure 2.11 : Voltage and Current Sensor

2.1.1.3 Autopilot Telemetry Hardware

Telemetry modem is used to transfer data between Ground Station and Autopilot via radio signals. The telemetry system plays a vital role while controlling the aircraft through a ground control station. The link between the PC and autopilot is carrying control signals while flying in manual mode. On the other and we can get all the information from aircraft to track it, to diagnose it or we can send new mission parameters or update the parameters during a flight. Telemetry systems are also shortening the development phase because they are acting as a diagnostic device by sending the parameters real-time to the ground station. RC receiver (Figure 2.12) is a one-way communication device. It transmits RC Transmitter Stick commands to the receiver. On the receiver there are 7 outputs generating PWM signals compatible with RC Servo hardware. This System is 1st priority Command source of the outopilot system. Regardless from RF modem (Figure 2.13) signal strength or availability, the RC receiver – Transmitter pair is getting commands from ground.

Figure 2.12 : RC Receiver

Figure 2.13 : RF Modem

2.1.1.4 Autopilot Actuators

RC Servos(Figure 2.14) are small servo motor with controller inside which are able to move their mechanical output shaft in desired direction and position by getting PWM signals from any signal source. RC servos are mostly controlled by 3 wires. Ground, V-Supply and Signal. Servo control modules can be external or internal. In this thesis an internal PWM generator is used to generate servo control signal. In most UAV systems the servos are closed loop systems, which only get an external position signal. According to the signal the servos are traveling rapidly to the desired position. In most cases of UAV development there is no need to get a feedback from servo motors. It is assumed that the desired position is always within an acceptable error range. Making this assumption makes the servo controlling an easy task for micro controller. The correct PWM signal generation is the only necessary task for

the servo controller. The position of servo arm and the PWM signal to be generated is explained in the figure 2.15.

Figure 2.14 : RC Servo

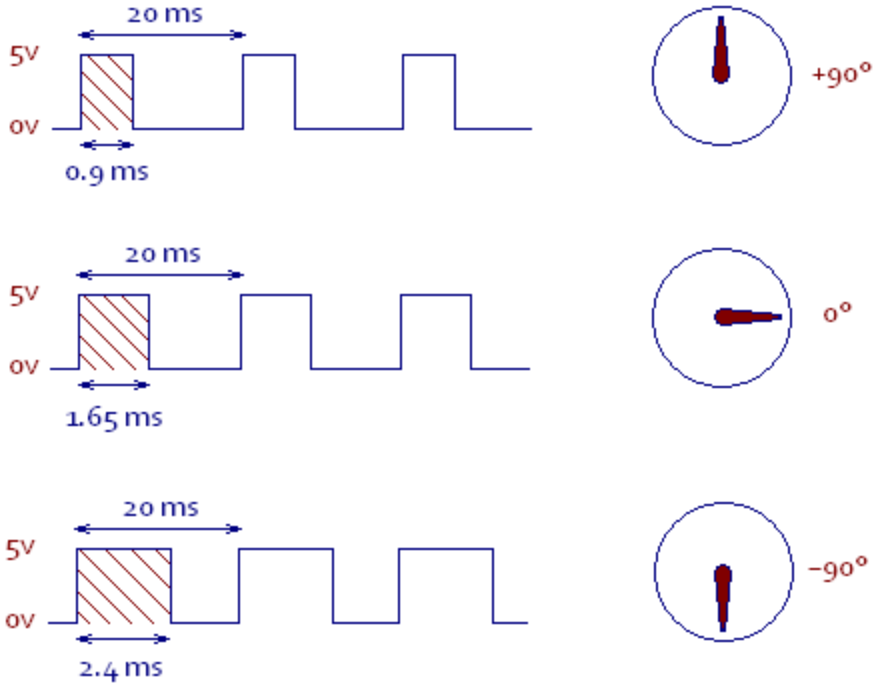

Figure 2.15 : RC Servo travel vs. PWM signal

2.1.2 Telemetry Software

Autopilot software is written in c++ language. The first thing is to make the autopilot hardware act as a gateway between actuators and RC receiver. In a standard RC aircraft the signals are sent from transmitter to receiver inside the aircraft and this receiver is directly connected to servos. In an UAV autopilot system, the autopilot is positioned between the RC receiver and Servos. The first code to be implemented should be reading the PWM signals from receiver and directing them to the servos through the digital output pins of controller. Below figure 2.16 describes the connection of autopilot betweenRC receiver and actuator.

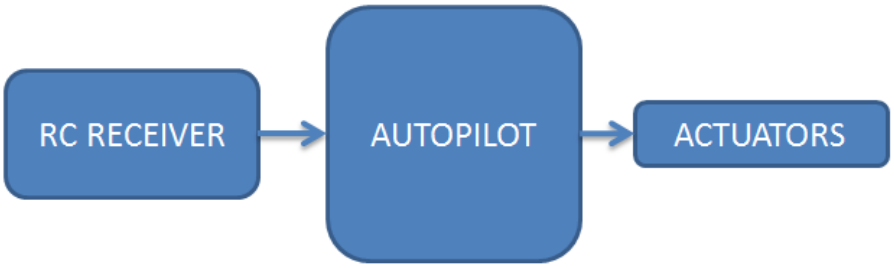

Figure 2.16 : RC Receiver , Autopilot and Actuators

Our controller has PWM output routines as standard but reading the PWM signals is a bit complex job. There has to be some interrupt vectors already set up in the code triggered by every rising edge and falling edge of the PWM signal. The duration between rising edge and falling edge determines the servo position command. This duration can only be captured using hardware interrupts. Such a hardware interrupt pseudo code based on C language can be implemented as follows.

```
OnRisingEdge()
{
lastrisingedge_ms = micros();
}
OnFallingEdge()
{
Servo_ms = micros()-lastrisingedge_ms;
```

Servo_pos=map(Servo_ms,900,2000,-45,45); //servo_ms 900 means -45 degree //servo armangle

}

In a normal operation of flight the plane is directly controlled by the RC Transmitter.(Figure 2.17) Commands are sent via sticks on the transmitter. The left stick stands for throttle and yaw command, the right stick is for roll and pitch movements of aircraft. There are also several switches which are acting like on-off commands. The lef top switch is used as Autopilot Enable switch. In the autopilot software the servo input signals can be converted to desired pitch or roll angle commands.

Figure 2.17 : RC Transmitter

The second thing is to build a reliable communication system between controller and ground station. To do that there has to be a data protocol to communicate efficiently.

The telemetry modem is sending data as byte packets. For binary communication the start and the end of the byte packets has to be defined. After receiving the packet I the packet has to be checked with the checksums to be sure if they are valid or not. Below there is a data protocol structure. The firs 3-4 bytes can be reserved as data prefix. Data prefix is a fixed group of byte with constant values indicating the message start. Data can be defined as long as wished. In this project the data length is always fixed for simplicity. The checksum bytes can be 2 bytes. It can be XOR checksum or just summation of data in two bytes. This is used to check the message validity. Below Table 2.1 shows the data structure of the communication between Autopilot- Ground Station and Autopilot – IMU.

Table 2.1 : Communication Protokol

byte 0	prefix
byte 1	
byte 2	
byte 3	data
byte 4	data
byte 5	data
byte 6	data
..	data
...	data
...	data
...	data
byte n-1	checksum a
byte n	checksum b

Autopilot system is programmed to send continuous and ask-answer type messages. Normally the autopilot system is sending messages of aircraft variables continuously. But some bytes in the protocol are reserved as 2 bytes message_id and 4 byte data which are only filled with meaningful data when a command from ground station is received. The message bytes are fixed and separated with a page id. Page id is a mechanism to be able to extend the message structure while continuing the development. Even the variables are reserving a start byte and length they are not sent in each packet. The page id for each packet is an incremental number going to zero after max page number is achieved. This way for the next generations of this autopilot system new variable from autopilot can be monitored by just adding new pages. The telemetry structure of Autopilot – Groundstation is given in Table 2.2

Table 2.2 : Autopilot Groundstation Protocol

byte no	def	page 1	page 2	page 3	page 4	page 5	page 6	page 7
0	prefix	101	101	101	101	101	101	101
1	prefix	102	102	102	102	102	102	102
2	prefix	103	103	103	103	103	103	103
3	hr_page_no	0						
4		roll						
5		roll						
6		pitch						
7		pitch						
8		yaw						
9		yaw						
10		roll dot						
11		roll dot						
12		pitch dot						
13		pitch dot						
14		yaw dot						
15		yaw dot						
16		control ael						
17		control ael						
18		control elev						
19		control elev						
20		control thro						
21		control thro						
22	mr_page_no	0	1	2				
23		gps heading	command_roll	gps_longitude				
24		gps heading	command_roll	gps_longitude				
25		gps heading	command_pitch	gps_longitude				
26		gps heading	command_pitch	gps_longitude				
27		gps altitude	command_thro	gps_latitude				
28		gps altitude	command_thro	gps_latitude				
29		gps altitude	ground_speed	gps_latitude				
30		gps altitude	ground_speed	gps_latitude				
31		airspeed	ground_speed	batt_volt				
32		airspeed	ground_speed	batt_volt				
33	lr_page_no	0	1	2	3	4	5	6
34		gps_numsats	rc3	distance_from_home	samplingrate	next_waypoint_latitude	statuscode	batterycurrent
35		gps_numsats	rc3	distance_from_home	samplingrate	next_waypoint_latitude	statuscode	batterycurrent
36		gps_quality	rc4	distance_from_home	next_waypoint_longitude	next_waypoint_latitude	statuscode	
37		gps_quality	rc4	distance_from_home	next_waypoint_longitude	next_waypoint_latitude	statuscode	
38		rc1	rc5	home_direction	next_waypoint_longitude	next_waypoint_altitude	batterymahcounter	
39		rc1	rc5	home_direction	next_waypoint_longitude	next_waypoint_altitude	batterymahcounter	
40		rc2	rc6	next_waypoint_direction	targetairspeed	next_waypoint_altitude	batterymahcounter	
41		rc2	rc6	next_waypoint_direction	targetairspeed	next_waypoint_altitude	batterymahcounter	
42		msg_id	msg_id	msg_id	msg_id	msg_id	msg_id	
43		msg_id	msg_id	msg_id	msg_id	msg_id	msg_id	
44		byte1	byte1	byte1	byte1	byte1	byte1	
45		byte2	byte2	byte2	byte2	byte2	byte2	
46		byte3	byte3	byte3	byte3	byte3	byte3	
47		byte4	byte4	byte4	byte4	byte4	byte4	
48	chka							
49	chkb							

As seen above , some group of variables have only 1 page which is named as “high rate page” some have 3 pages named as “middium rate page” some has 7 pages named as “low rate page”. The reason for that is to group the refresh rate of variables. Some variables are need to be sent more frequent then the others to the ground software. For example the gps position is updated 5 times in second. There is no reason to update it more frequent. But the attitude of the aircraft is updated 50 times per second. This information can be sent as frequent as possible to allow real-time graphical attitude tracking.

2.1.3 Aircraft Control Software

Next topic will cover detailed algorithms for aircraft control. The notations in Table 2.3 have been used for all types of PID controllers implemented in the autopilot code.

Table 2.3 : Autopilot PID loop notation

Variable Name	Description
Kp_xxx	Proportional gain of xxx controller
Kd_xxx	Derivative gain of xxx controller
Kdd_xxx	Second derivative gain of xxx controller
Ki_xxx	Integral gain of xxx controller
Kff_xxx	Feed forward controller gain of xxx controller
Hi_saturate_xxx	Highest possible output value
Lo_saturate_xxx	Lowest possible output value
Ki_xxx_limit	Positive and negative accumulation limit of integral control

2.1.3.1 Attitude Stabilisation Code

The first part to implement an Autopilot System Software is to write the Attitude Stabilization code. For attitude control a PID controller with Feed – Forward addition is implemented. Controller software is taking the angular position and rates from IMU and sending proper signal to the servos which are connected to ailerons and elevator. Figure 2.19 shows the general control diagram of attitude stabilization code.

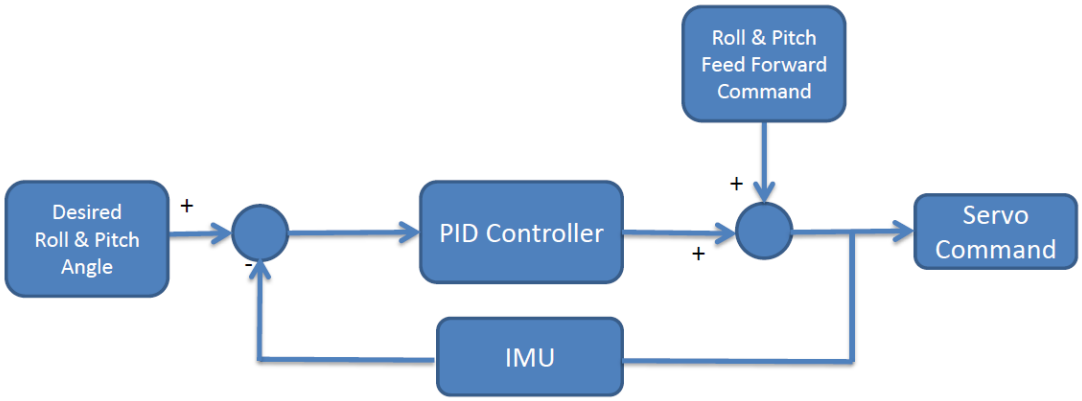

Figure 2.18 : Attitude control diagram

This controller can be activated and deactivated during flight. RC transmitter stick movements are taken as desired roll and pitch angles. The aircraft should maintain the desired angles. (Figure 2.19) .

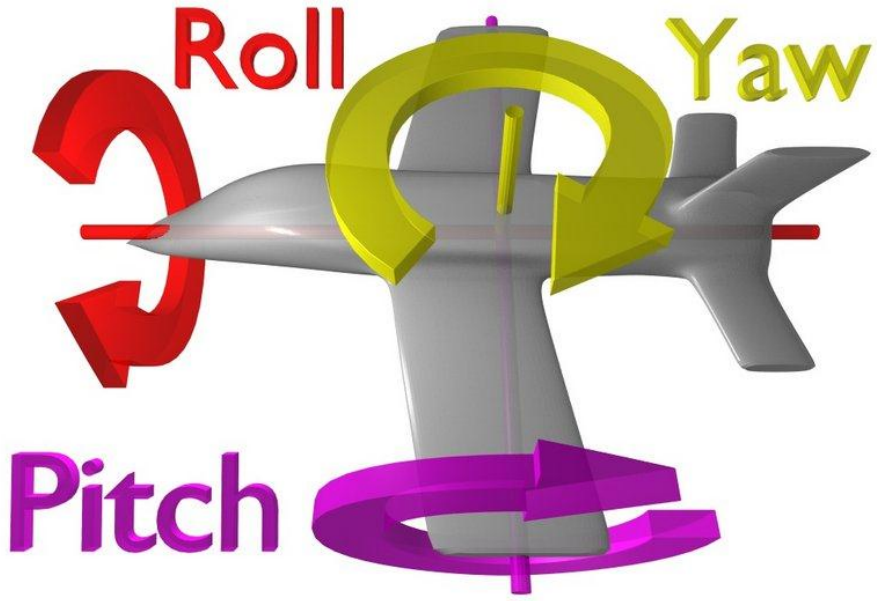

Figure 2.19 : Euler Angels

All other controllers are orienting the aircraft by changing the desired roll and pitch angle variables. These variables can be changed not only by direct user input but also, by waypoint controller or heading controller or altitude controller. Variables in the Table 2.4 are processed by the controller.

Table 2.4 : Attitude Control Variables

Variable Name	Description
steadyalpha	Alpha angle for stedy flight
steadyrolldrift	Roll offset of IMU
Kp_pitch	
Kd_pitch	
Kdd_pitch	
Ki_pitch	
Kff_pitch	
Hi_saturate_pitch	
Lo_saturate_pitch	
Ki_pitch_limit	
Kp_roll	
Kd_roll	
Kdd_roll	
Ki_roll	
Kff_roll	
Hi_saturate_roll	
Lo_saturate_roll	
Ki_roll_limit	

Attitude control pseudo code is given as:

control_aile=int(mapf(saturatef(staticpressurecorrection()*(Kff_roll*command_roll+Kp_roll*rollerror+Kd_roll*rollerrordot+Kdd_roll*rollerrordotdot+Ki_roll*rollerroraccumulator),Lo_saturate_roll,Hi_saturate_roll),-45,45,-500,500));
control_elev=int(mapf(saturatef(staticpressurecorrection()*(Kff_pitch*command_pitch+Kp_pitch*pitcherror+Kd_pitch*pitcherrordot+Kdd_pitch*pitcherrordotdot+Ki_pitch*pitcherroraccumulator),Lo_saturate_pitch,Hi_saturate_pitch),-45,45,-500,500));

2.1.3.2 Heading Stabilisation Code

Heading stabilization is code is a very important code to maintain waypoint navigation. The aim of this code is to orient the aircraft to desired heading by changing the desired roll angle and feeding this information to attitude controller code. The control diagram for heading stabilization is given in figure 2.20

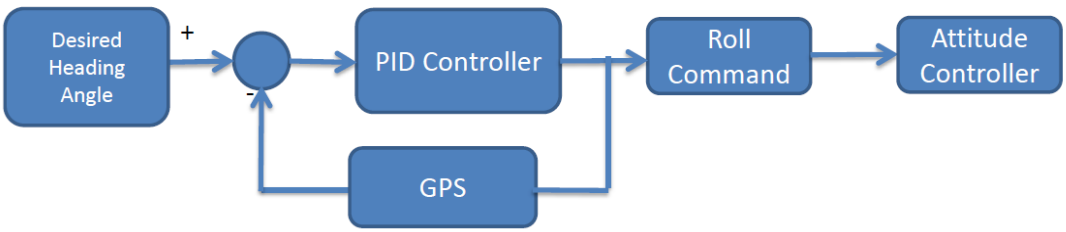

Figure 2.20 : Attitude Control Variables

Heading controller is mostly controlled by waypoint controller. GPS can feed reliable heading information above ground speeds of 10km/h. If there is a headwind and the aircraft has zero or negative groundspeeds then the GPS is no more reliable heading sensor. The heading angle 0 means North Pole heading. 180 means South Pole heading. Aircraft heading via GPS is calculated internally in GPS hardware. This calculation is made by finding bearing of two points in the time and it may not always be the true heading of the aircraft but always gives the flight path heading. Therefore this kind of data gives better results than compass heading data since the aircraft may not always be oriented to the direction it flies. Variables in Table 2.5 are processed by the controller.

Table 2.5 : Heading Control Variables

Kp_heading
Kd_heading
Ki_heading
Kff_heading
Hi_saturate_heading
Lo_saturate_heading
Ki_heading_limit

The pseudo code for the heading control is given as:

command_roll=saturatef(Kp_heading*headingerror+Ki_heading*

headingerroraccumulator, Lo_saturate_heading, Hi_saturate_heading);

2.1.3.3 Alitude Stabilisation Code

An autopilot should be able to maintain desired altitude during autonomous flight. To do that altitude information from the sensors are needed. In this work this altitude information is taken from both GPS and barometric pressure sensor. The Altitude data of GPS may vary while flying even if the aircraft is maintaining the altitude. The barometric pressure sensor may have better results during flight but it even it may be affected by atmospheric conditions. The aim of this controller is to make the altitude error zero. Desired altitude command is fed by waypoint controller or direct stick inputs. Error is defined as desired altitude – current altitude. Figure 2.21 describes the controller used for altitude stabilisations.

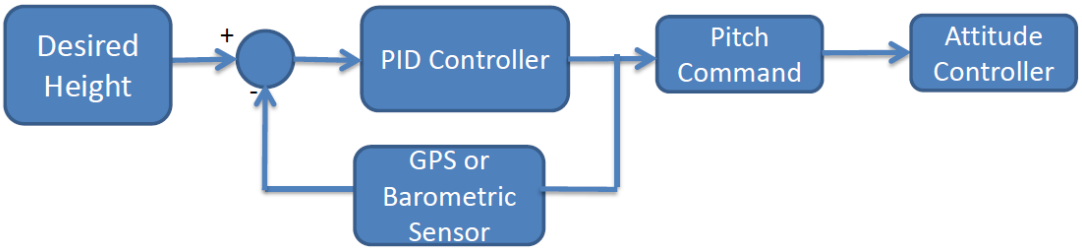

Figure 2.21 : Attitude Control Variables

Variables in Table 2.6 are processed by the controller.

Table 2.6 : Altitude Control Variables

Kp_altitude
Kd_altitude
Ki_altitude
Kff_altitude
Hi_saturate_altitude
Lo_saturate_altitude
Ki_altitude_limit

Altitude control pseudo code is given as:

command_pitch= saturatef (Kp_altitude * altitudeerror / 1000 + Ki_altitude * altitudeerroraccumulator, Lo_saturate_altitude, Hi_saturate_altitude);

2.1.3.4 Airspeed Control Code

It is essential to control the aircrafts airspeed during autonomous flight. The aircraft may change its orientation all the time and this is affecting the aircrafts air speed. There are other effects like wind gust, flight direction change. If the aircraft cannot maintain the airspeed, it may stall or aircrafts airspeed may increase its speed beyond its design limits. The proposed controller is a feed forward PID controller, taking the airspeed value from aircrafts pitot tube sensor and controlling the throttle input. Proposed control diagram is showm in Figure 2.22

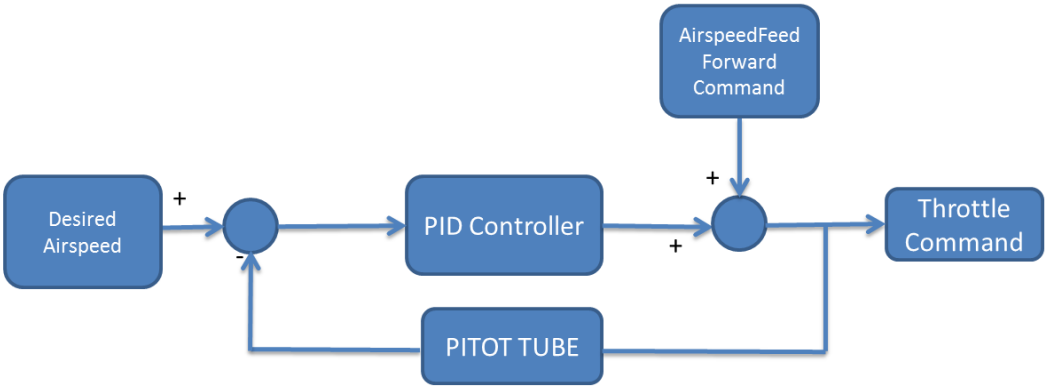

Figure 2.22 : Airspeed Control Variables

Table 2.7 shows the variables processed by the controller.

Table 2.7 : Airspeed Control Variables

Kp_airspeed
Kd_airspeed
Ki_airspeed
Kff_airspeed
Hi_saturate_airspeed
Lo_saturate_airspeed
Ki_airspeed_limit

Airspeed control pseudo code is given as :

control_thro=int(cmdoff3+saturatef(Kff_airspeed*command_thro + Kp_airspeed*airspeederror,Lo_saturate_airspeed,Hi_saturate_airspeed)*10);

2.1.3.5 Cross Track Controller

In a waypoint navigation control algorithm it is essential not only to go to next desired geographical location but also track the path between two waypoints. Figure 2.23 explains the described problem.

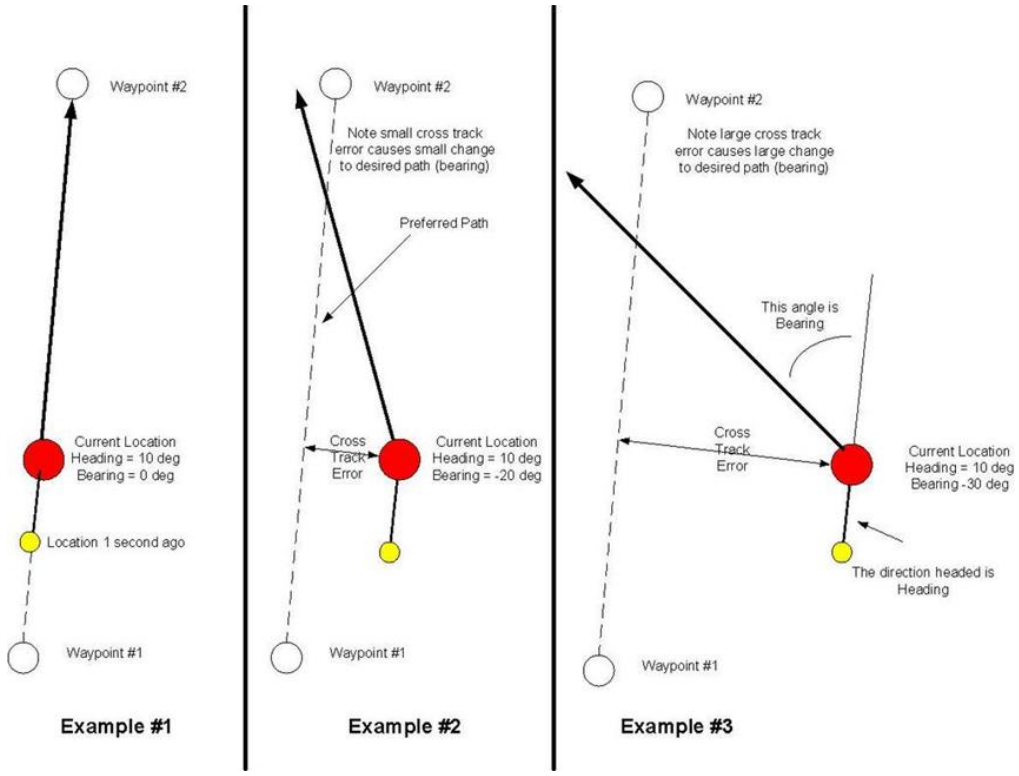

Figure 2.23 : Cross Track Error Description

The controller should calculate the desired bearing of the aircraft to the next waypoint and at the same time add more bearing if there is a cross track error. The amount of additional bearing command can be parameterized as a PID controller. The amount of distance can be multiplied by factor Kp_crosstrack and fed in to equation of bearing to force the aircraft bear more to keep the desired path. In our example the cross track is only fed as Proportional controller with saturation limits. That means the aircraft bearing to the next waypoint can only be increased by a defined limit. The proposed controller shema is given in figure 2.24.

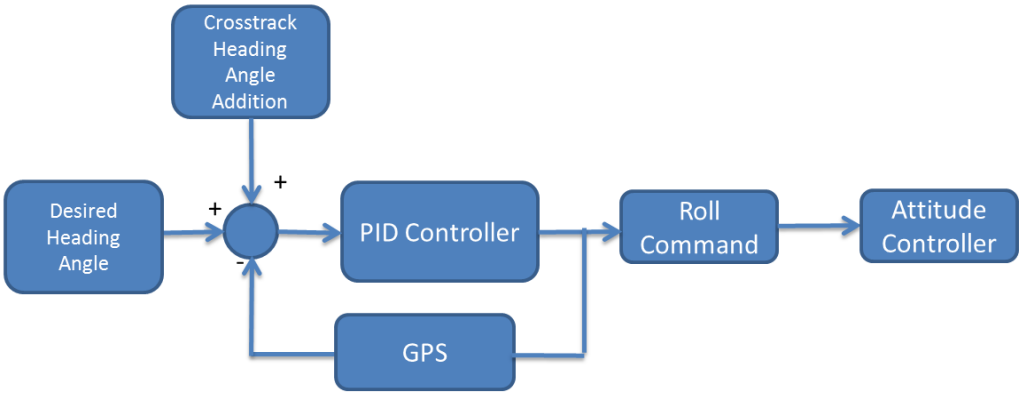

Figure 2.24 : Cross Track Error Controller

Above controller is the same as heading controller with an addition of crosstrack heading addition. Crosstrack heading addition can be calculated in a PI controller as crosstrackerror variable input. PI controllers I parameter has a very good positive effect on keeping the track within desired error range.

2.1.3.6 Waypoint Navigation

Waypoint navigation is the most important task which an autopilot system should complete. Waypoint navigation must use all the control algorithms like attitude stabilization, heading stabilization, altitude stabilization, and airspeed control and cross track error correction. A waypoint task consists of 3D geographical coordinates, air or ground speed between coordinates. The aircraft should go to desired geographical coordinates at desired altitude and with a desired air or groundspeed. Figure 2.25 describes the proposed controller for waypoint navigation.

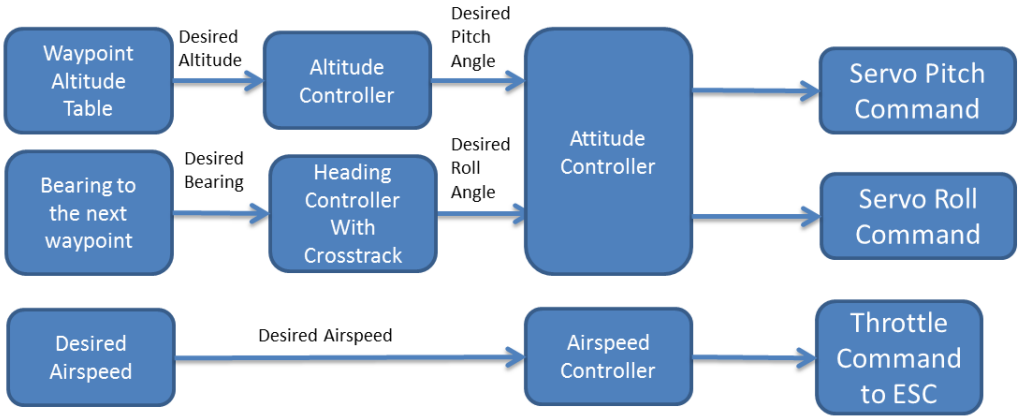

Figure 2.25 : Waypoint Navigation Controller

As seen above, the waypoint controller is using all the other controllers to complete the waypoint tasks. The waypoint controller should calculate the desired bearing by every update of GPS sensor. To calculate the desired bearing following formula has been used:

bearing=atan2(sin(lon2-lon1)*cos(lat2),(cos(lat1)*sin(lat2))sin(lat1)*cos(lat2)* cos(lon2-lon1)));

This pseudo bearing calculation code calculates the bearing angle of 2 given geographical coordinates as latitude and longitude. Another calculation which has to be made by the autopilot is to calculate distance between the aircraft and the next waypoint. If this distance is lower than a predefined value (waypoint_accept_distance) , the autopilot next waypoint index is increased by 1. The distance can be redefined from ground station. The default value for this is 30m.

The distance between 2 given geographical coordinates can be calculated with the following formula:

dlon = dtor(lon2 - lon1);

dlat = dtor(lat2 - lat1);

a = pow(sin(dlat/2),2) + cos(dtor(lat1)) * cos(dtor(lat2)) * pow(sin(dlon/2),2);

c = 2 * atan2(sqrt(a), sqrt(1-a));

dist = 6378140 * c;

2.1.3.7 Main Control Code

Main control code structure can be seen in figure 2.26

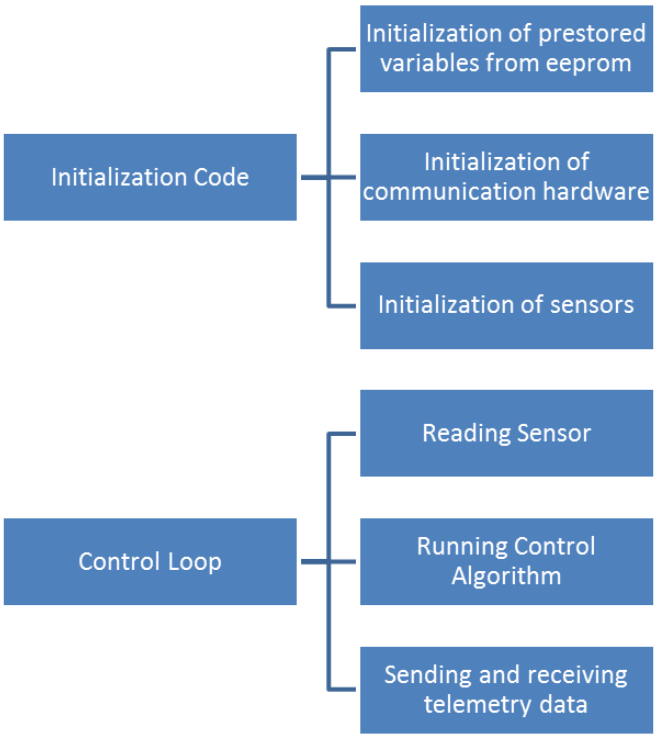

Figure 2.26 : Main Control Code Diagram

As seen above, the main controller is running in a control loop. It is not only responsible from control tasks but also responsible of hardware sensor readings and telemetry tasks. To be able to do all these work efficiently, calculation economy must always be taken in to account. As many as possible amount of calculations has to be made by the external controllers like IMU controller and GPS controller. The current controller is able to perform control calculations at 120Hz. This calculation speed is more than the speed of IMU.

2.2 Ground Station

Ground station is a base station where the actions of flying platform can be monitored and commands are sent to the aircraft. We can split the ground station in to two main components. Figure 2.27 shows the ground station component structure.

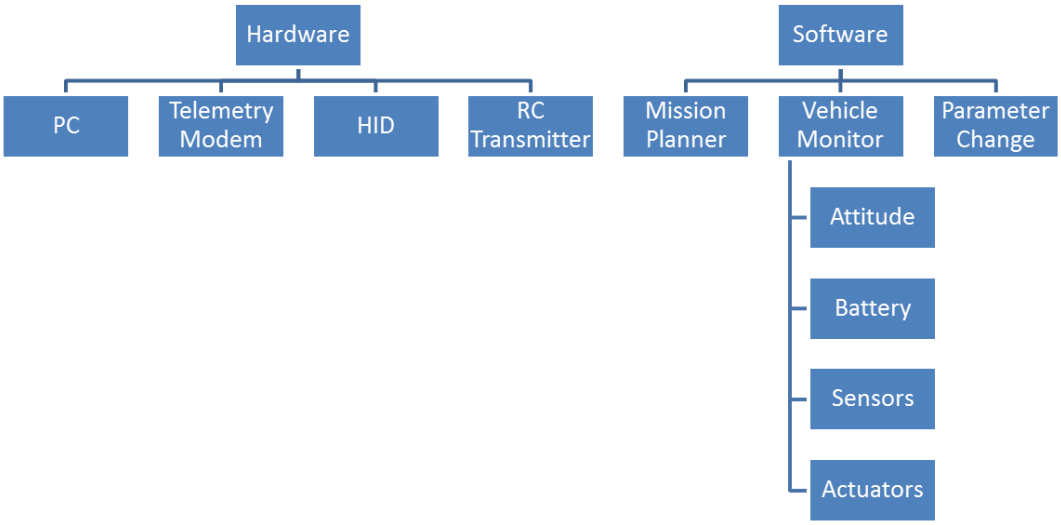

Figure 2.27 : Ground Station Components

2.2.1 Ground Station Hardware

Ground station hardware structure consists of PC, telemetry modem, human interface device and telecommunication system which is a RF modem in this case.

2.2.1.1 Ground Station PC

PC hardware is a windows platform running special software as ground station. PC is connected to telemetry modem, and joystick. Therefore 2 usb ports are needed. The ground station software needs at least 1920x1080 pixels resolution. Below figure 2.28 shows the ground station pc which is used in this thesis.

Figure 2.28 : Ground Station PC

2.2.1.2 Telemetry Modem

Telemetry modem is the same type as you can find in the aircraft which is capable of both sending and receiving data. The data transmission rate as RF is 19.200bps fixed as factory default. The communication between modem and PC hardware is made a standard USB cable. Below figure 2.29 shows the RF modem used in ground station for communication.

Figure 2.29 : Ground Station Telemetry Unit

2.2.1.3 HMI Joystick

Joystick is a human interface device to control the aircraft. It is a secondary control methot apart from RC transmitter. With joystick, roll, pitch , yaw and throttle commands can be sent. The buttons can be configured to perform other tasks in the autopilot like camera gimbal control, parachute drop etc. Joystick commands are using the telemetry modem to send commands to the autopilot. Therefore it has more channels than a standard RC Transmitter which has only 6 different servo channels. Bu the reliability of the joystick commands are pure. In case of any failure in PC hardware (program not responding, blue screen etc.) the communication of aircraft with ground station is lost. That means the joystick commands cannot be sent to the aircraft. In such case there is always a backup system which is RC transmitter. The autopilot software is always getting the RC signals from RC transmitter. If the RC transmitter is disabling the autopilot mode the RC transmitter direct stick control inputs are valid. This is a safety program which is implemented inside the autopilot. Below Figure 2.30 shows the joystick which is used in this thesis experiments.

Figure 2.30 : Joystick

2.2.1.4 RC Transmitter

RC transmitter is the primary control input device. It is always on but can temporary move the authority to autopilot by toggling a switch on top left of the device. In this project, a standard RC transmitter has been used. 6 Channels of the transmitter are active and sending signals to the RC receiver inside the aircraft. Every stick movement is generating a proper PWM signal from the RC Receiver. These signals are directly connected to the autopilot. Below figure 2.31 shows the RC transmitter and stick movements.

Figure 2.31 : RC Transmitter

Above you can see the stick descriptions according to MODE 2 layout. With RC transmitter there is always a possibility to manually fly the aircraft for security reasons. In case of failure in the aircraft or parameter misadjustments the pilot can easily realize that something goes wrong and disable the autopilot and move to manual flight immediately.

2.2.2 Ground Station Software

Ground station software is written by Delphi 7.0 programming language. Several programming skills like: Database Programming, COM and JavaScript, Google Maps API, Communication Protocol Development have been used (Cantu, 2003). This software has been designed from scratch specifically to meet the ground station software needs. Ground station software has a built in mission planner module, vehicle monitor unit and parameterization interface where the operator can change the control parameters of the autopilot even during flight. Figure 2.32 shows the main interface of ground control station software.

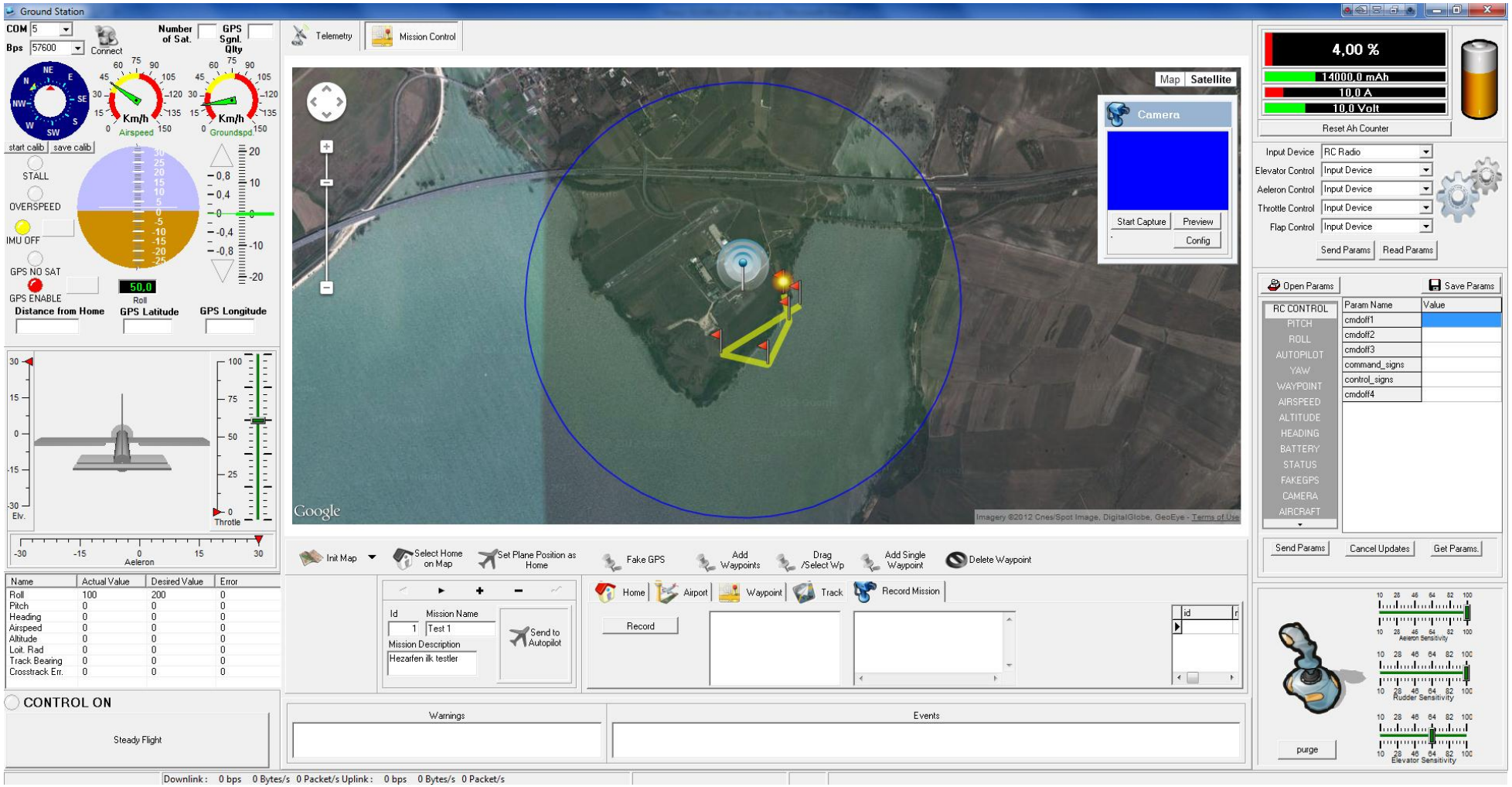

Figure 2.32 : Ground Control Station

2.2.2.1 Mission Planner

Mission planner of ground software is designed to allow user select points on map and upload the data to the aircraft's autopilot system via telemetry modem. Mission planner is designed also as a database system where you can insert delete or alter missions. A mission table is holding the data of home coordinates of mission, waypoint lists with altitudes, mission name and mission description. This allows user to select predefined flight area easily from database navigation menu. Below figure 2.33 shows the mission planner UI.

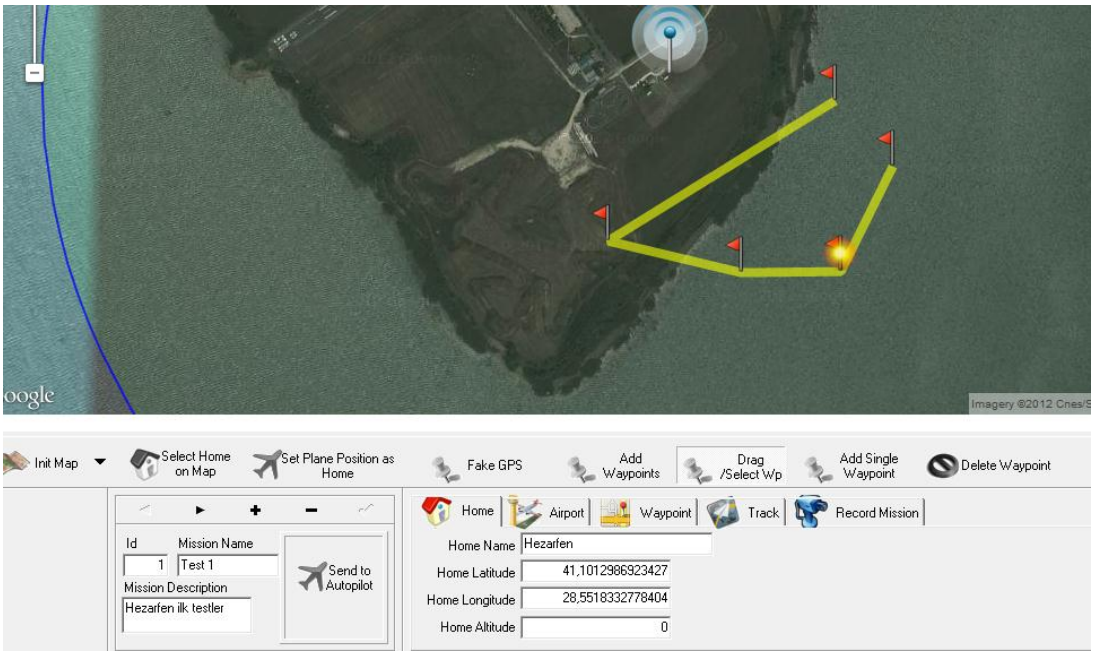

Figure 2.33 : Mission Planner UI

As seen above, mission planner has a main map window which is using the Google Maps API V3, which is specially integrated for this ground station software using COM, and JavaScript based programming. In this map, it is possible to add, delete or move waypoints. Home point definition via map is also allowed. "Send to Autopilot" button is sending all the waypoint information to the autopilots RAM. After sending the waypoint data, the aircraft can follow the waypoint instruction via activation button on RC Transmitter.

2.2.2.2 Vehicle Variable Monitor

It is very important to monitor the vehicle during the flight. Program can monitor v Attitude, Sensors, Battery, Actuators, Automatic Control Variables. Figure 2.34 shows the attitude and sensor monitoring capability of ground station software.

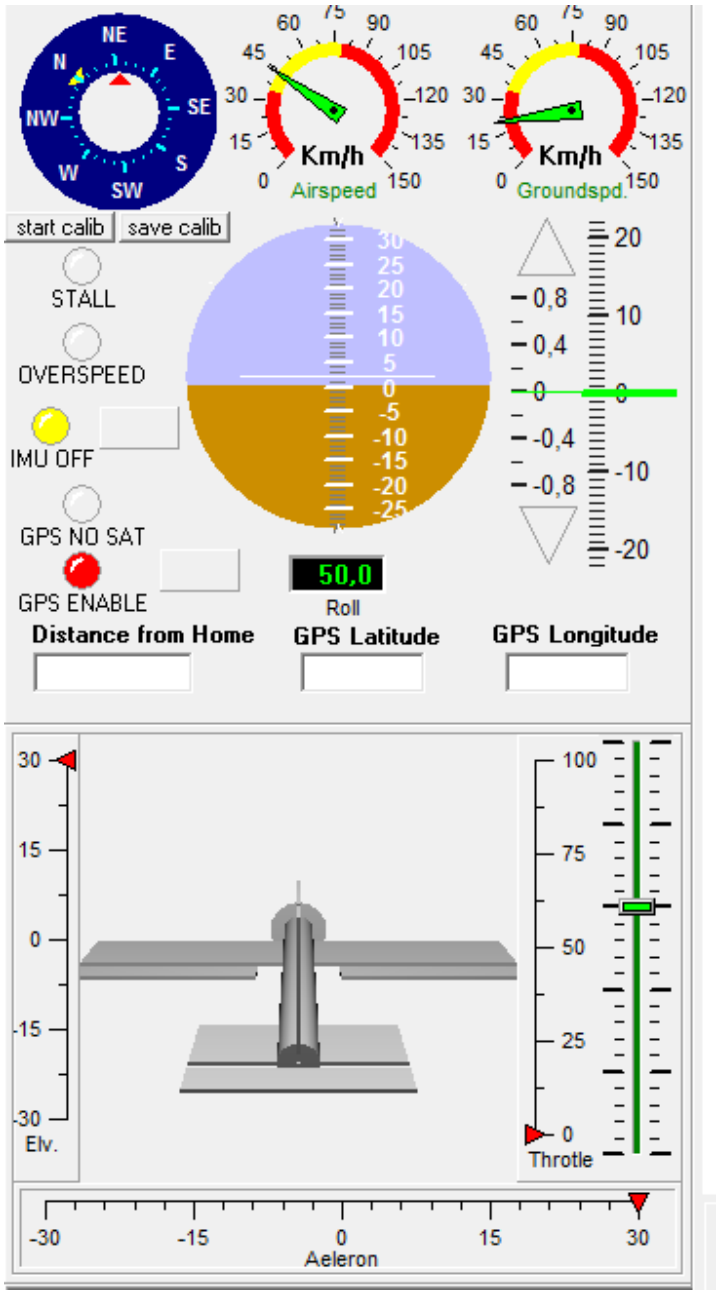

Figure 2.34 : Attitude Monitor

ttitude monitor section of the program is giving detailed attitude data about the aircraft. It is also easy to monitor all the control surfaces of the aircraft during flight. IMU and GPS sensor status can be monitored as well. Battery monitoring is a very important task for UAV user. The whole system is depending on battery power on

electric UAV's. As seen above Voltage, Current and Energy parameters can e monitored during flight. There is also a percentage gauge which is calculating remaining energy in the battery with a method called coulomb counting. These calculations are made within autopilot hardware and sent through telemetry in to the ground station. When new battery is inserted Reset Ah Counter button has to be pressed once to reset the Ah counter. The initial capacity is given as a fixed parameter in parameter window. Below figure 2.35 show the battery monitoring sectionin detail.

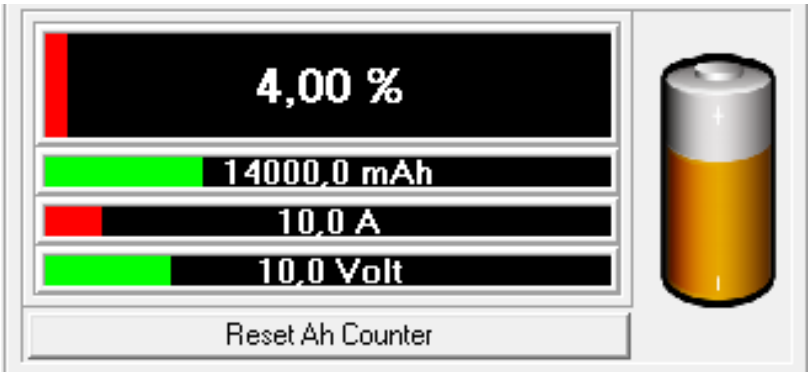

Figure 2.35 : Battery Monitor

During parameter settings of a new developed aircraft it is essential to monitor the input and output variables and the error which has to be zeroed with good PID control parameters. Below figure 2.36 shows the PID control input output monitoring section of software. Control on led indicates that the autopilot switch on the transmitter is toggled.

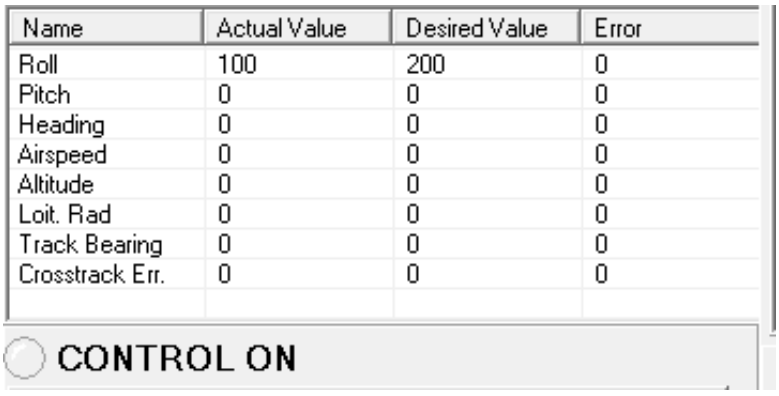

Figure 2.36 : PID Control Input , Output and Error Monitor

There is also a piloting tool in the ground station control software. Below you can find the piloting tool for several sensors and control variables. This tool makes it easy to diagnose sensor failures or noise in the sensors. All the IMU outputs, Barometric pressure sensor outputs can easily be monitored with this tool. Figure 2.37 shows the piloting toolof GCS.

Figure 2.37 : Pilotting Tool

2.2.2.3 Vehicle Variable Monitor

With the help of parameter change window, 109 different aircraft variables can be changed during flight. As seen in figure 2.38, parameters are grouped and can easily be accessed pressing the group button. There are 109 parameters, which can be changed during flight. Parameters can be stored by pressing Save Params button and retrieved back for later use by pressing Open Params button. Send params button sends all the changed parameters to outopilot. Cancel updates button cancels all the updated parameters from last Send Params button press or Power On action.

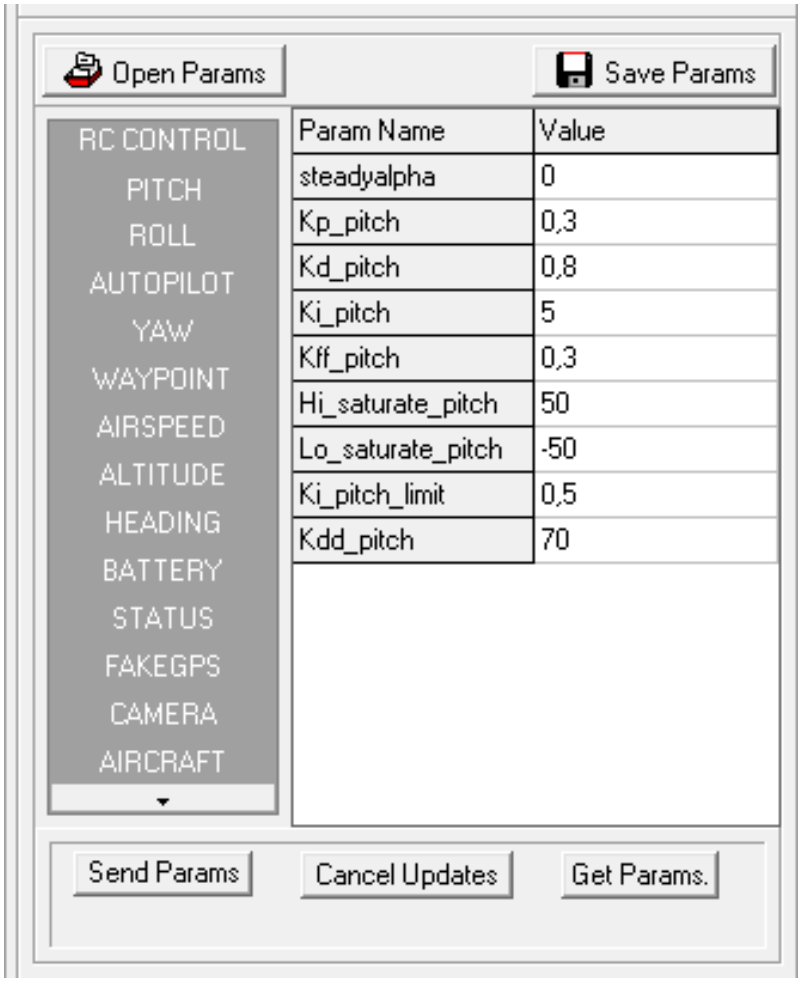

Figure 2.38 : Parameter Change Window

It is very important to monitor the vehicle during Mission planner of ground software. Another way of changing special flight parameters is control mode section usage. Control mode selection mechanism is there to control the actions if the user is toggling the switch from RC Transmitter. Figure 2.39 shows the easy parameterchange mechanism for control mode selection.

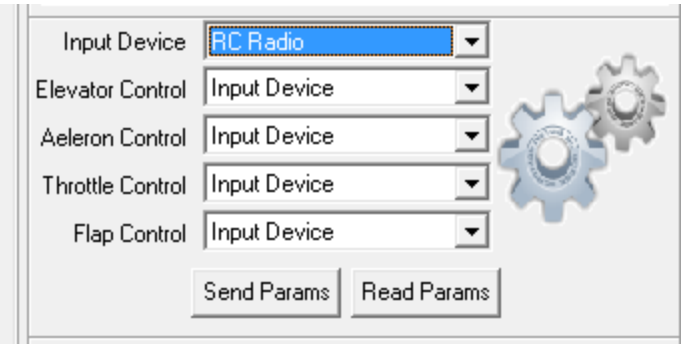

Figure 2.39 : Control Mode Selection

As stated before, RC transmitter is the primary control source. The autopilot behavior after control on switch is defined by control mode selection. This feature is very useful when first working with an aircraft. The control parameters can be set freely and then after activation the aircraft is observed. Input Device: Main command source can be defined as Joystick or RC Radio. If control mode is selected as "input device" in one of the control devices, the aircraft gets direct input from control device. Elevator Control: There are 3 different elevator control modes which are :Fixed Altitude, Flight Stabilizer, Waypoint. Fixed altitude control means that the aircraft tries to maintain the altitude when the control switch is activated via the RC transmitter. Flight Stabiliser Control means that the aircraft tries to keep his attitude at a constant pitch angle. Waypoint Control means the elevator tries to keep tha altitude according to give waypoints. Aileron Control: There are 3 different aileron control modes like Fixed Heading, Flight Stabilizer, Waypoint. Fixed heading control means that the aircraft tries to maintain its heading by a given heading angle. This feature is useful when heading parameters are first set. The aircraft reaction can be observed upon control activation. Flight stabilizer is there to stabilize the aircraft at roll axis. Waypoint control is controlling the ailerons so that the aircraft bears to the next waypoint. Throttle Control: Throttle control mode has 2 options like Speed Control and Waypoint Control. Speed control is controlling the aircrafts airspeed at a constant value. This value can be redefined by the parameter window. Waypoint control mode means that the aircrafts speed will be maintained by the table of waypoints.

3. FLIGHT EXPERIENCE

Several flights with the autopilot system have been performed. To test the hardware, a flying RC aircraft was needed. This aircraft is chosen as capable of carrying 200gr of payload. All the sensors are attached to the aircraft via duct tape or 2 sided tape. To maintain the stability the center of gravity is kep as in original configuration. The aircraft is phoenix with pusher propeller configuration. Figure 3.1, Figure 3.2 shows the airplane equipped with the controller just before hand launch.

Figure 3.1 : Airplane Handlaunch - 1

Figure 3.2 : Airplane Handlaunch - 2

The aircraft has a wingspan of 920mm and a TOW of 900gr. Waypoint flight tests have been successfully performed. Figure 3.3, 3.4 and 3.5 is showing succesfull waypoint navigation screenshots.

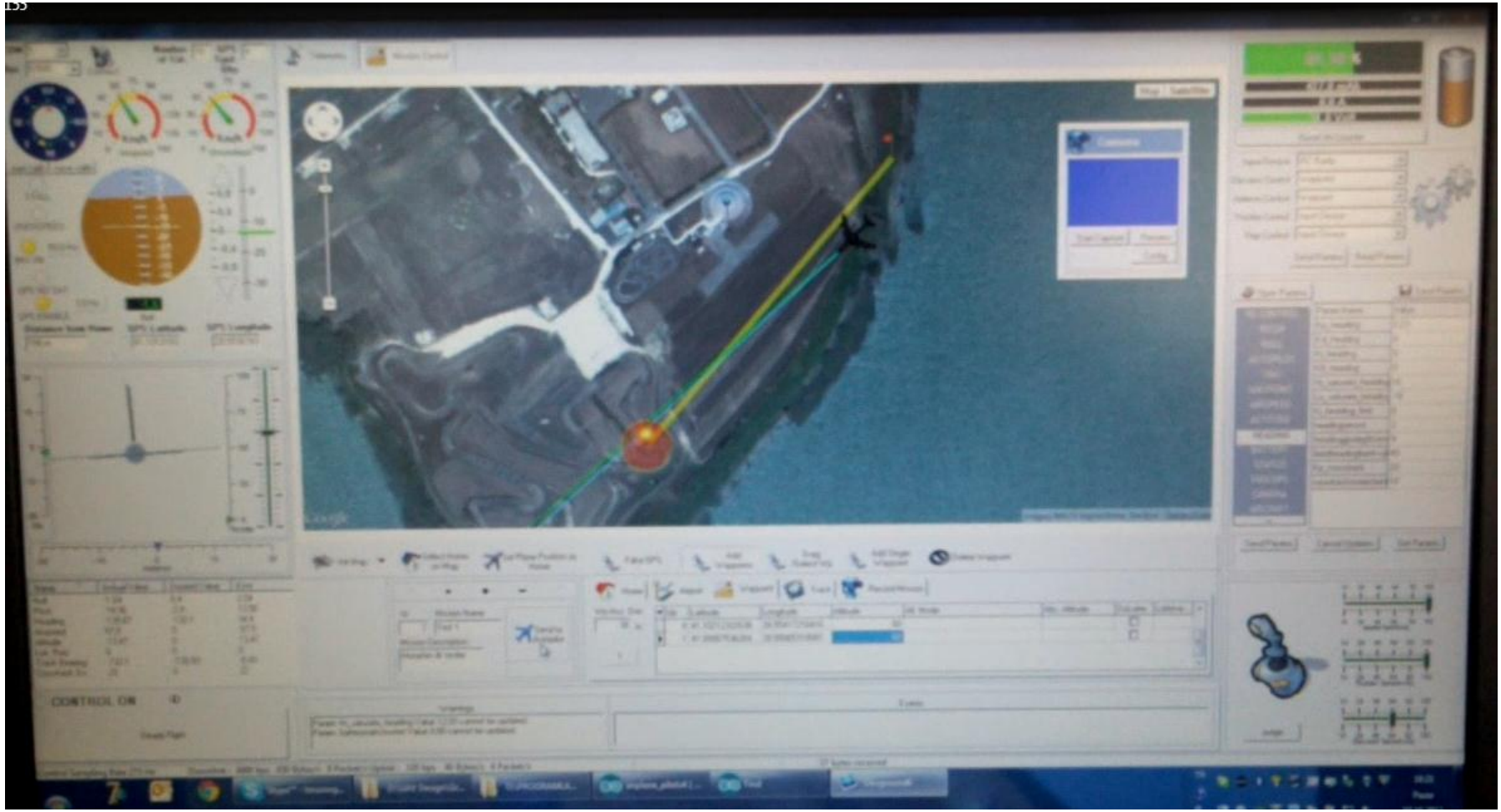

Figure 3.3 : Waypoint Screenshots -1

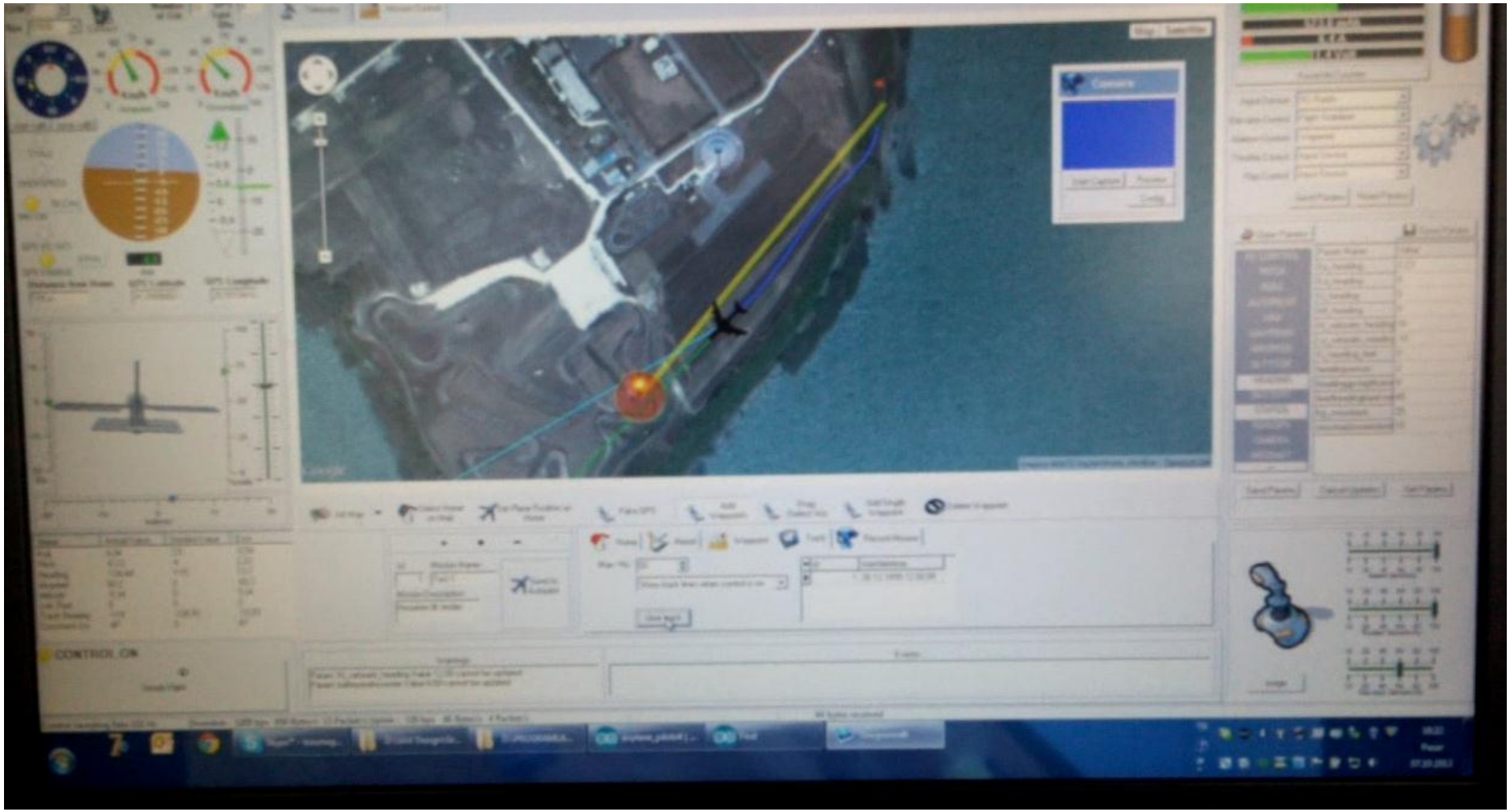

Figure 3.4 : Waypoint Screenshots -2

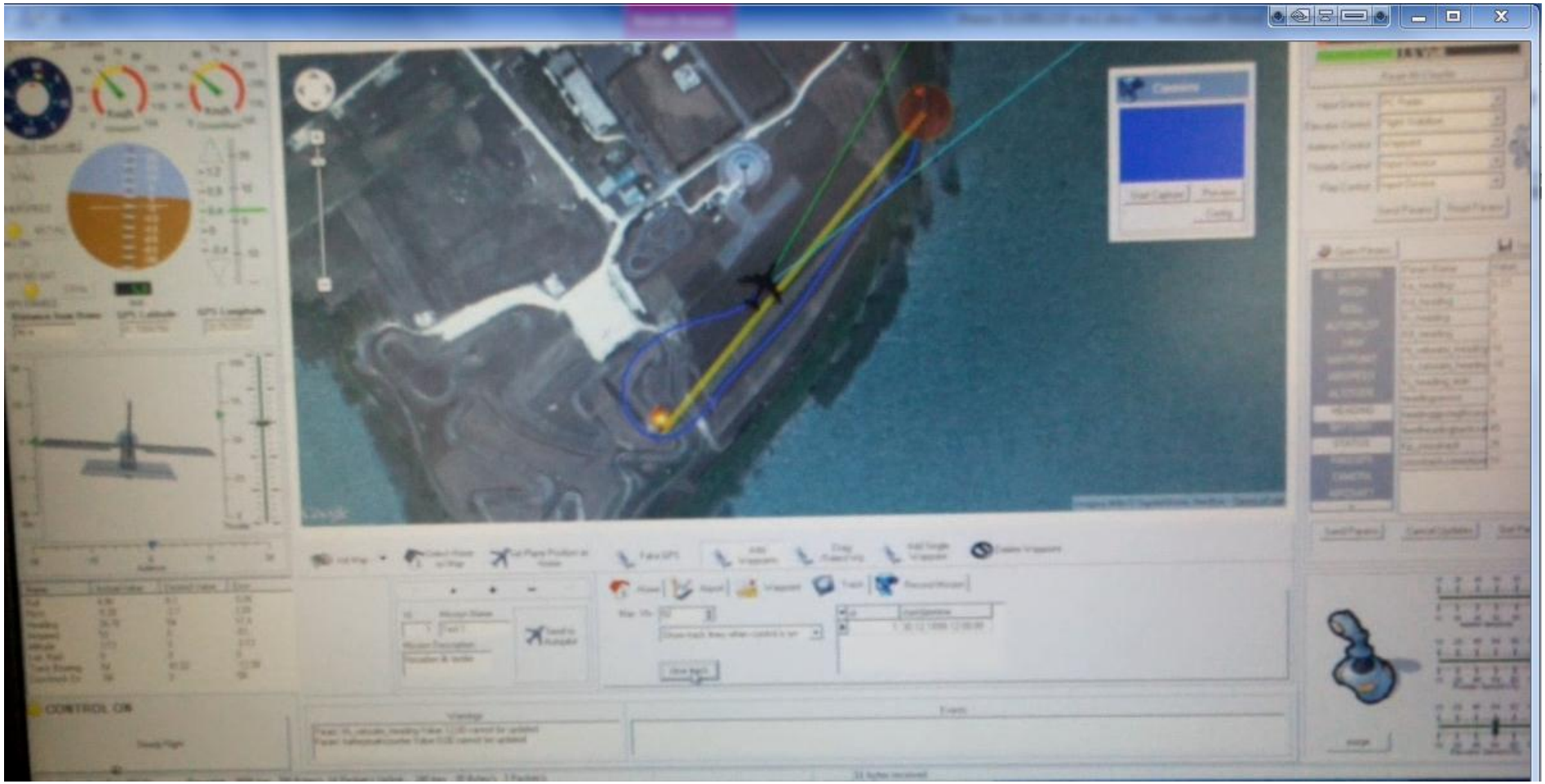

Figure 3.5 : Waypoint Screenshots -3

Above waypoint task consists of 2 waypoints repeating from first to second. As seen above, the aircraft has to additional lines in fron showing the current heading and desired heading.

4. CONCLUSION

The autopilot system has been tested successfully. Plane completed a 2 point waypoint task. The developed system is able to perform easy autonomous tasks. The most important part of this system is the telemetry hardware and software. With this system, the autopilots control parameters can be changed remotely. This mechanism allows the system to be integrated in to all kind of autonomous systems. The control code given in appendix can be changed according to aircraft type and system can be integrated into almost every type of aircraft.

REFERENCES

Albaker B.M. and Rahim N.A. (2011) . Flight path PID controller for propeller-driven xedwing unmanned aerial vehicles. *International Journal of the Physical Sciences* Vol. 6(8), pp. 1947–1964.

Bennett S. A. (1984) Nicolas Minorsky and the Automatic Steering of Ships. *Control Systems Magazine, IEEE Vol.4*

Cambone S. A. and Krieg K.J. and Wells L. (2005) 'USA's Unmanned Aircraft Roadmap, 005-2030,National Defense.

Cantù, Marco (2003) Mastering Delphi 7, Sybex Inc. New Jersey.

Nelson , Robert, Flight Stability And Automatic Control, WCB/McGraw-Hill, Ohio.

Pettersen , K. Y. and Brhaug .E. and (2005) Cross-track control for under actuated autonomous Vehicles. 44th IEEE Conference on Decision and Control, and the European Control Conference 2005

Sullivan, J.M. (2006). ''Evolution or revolution? The rise of UAVs," IEEE Technology and Society Magazine,vol. 25, no. 3, pp. 43-49.

Url-1 < https://developers.google.com/maps >, date retrieved 29.06.2011.

APPENDICES

APPENDIX A: Aircraft Control Code

APPENDIX A

```
void control_airplane()
{
     roll=ang_x-steadyrolldrift;//(int(ang_x)*1+rollold*2)/3;
     pitch=ang_y-steadyalpha;//(int(ang_y)*1+pitchold*2)/3;
     yaw=ang_z;
bitWrite(statuscode,0,switch_on);  //control off
if(switch_on){

   if(oldswitch_on==false) //rising edge
       {
         rollerroraccumulator=0;
         pitcherroraccumulator=0;

        if
((elevatorcontrolmode==1)||(elevatorcontrolmode==3)){next_waypoint_altitude=GPS_Altitude()+fi
xedaltitudeascendvalue*1000;altitudeerroraccumulator=0;}  //switch le beraber o anki
yüksekliğe bak ve aklında tut
        if
((aeleroncontrolmode==1)||(aeleroncontrolmode==3)){next_waypoint_direction=ang_z*100+fixedhea
dingbankvalue*100;headingerroraccumulator=0;}  //switch le beraber o anki heading e bak ve
aklında tut
        if(aeleroncontrolmode==4)  //control açılınca waypoint modunda ise bulunduğu konumu
başlangıç olarak al gidilecek koordinatlara bak yönelme ve yüksekli hesaplamaları yap
          {
            next_waypoint_id=0;
            next_waypoint_longitude=wplon[next_waypoint_id];
            next_waypoint_latitude=wplat[next_waypoint_id];
            target_waypoint_altitude=wpalt[next_waypoint_id];

last_next_waypoint_distance=CalcDistance(float(GPS_Lattitude())/10000000,float(GPS_Longitude(
))/10000000,float(next_waypoint_latitude)/10000000,float(next_waypoint_longitude)/10000000);
            last_waypoint_altitude=wpalt[next_waypoint_id];//GPS_Altitude();
controlaçıldığında ilk wp yüksekliğine git

trackbearing=CalcBearing(float(GPS_Lattitude())/10000000,float(GPS_Longitude())/10000000,floa
t(next_waypoint_latitude)/10000000,float(next_waypoint_longitude)/10000000); //şu anki konum
ile bir sonraki waypoint arasındaki çizginin kuzey çizgisi ile yaptığı açı
            //trackbearing=18000;
          }
       }


                  if(controlcounter%30==0){ home_direction=(
CalcBearing(float(GPS_Lattitude())/10000000,float(GPS_Longitude())/10000000,float(home_latitu
de)/10000000,float(home_longitude)/10000000));}

     if ((elevatorcontrolmode!=2)&&(elevatorcontrolmode!=0))
         {
            if(elevatorcontrolmode==3){next_waypoint_altitude=next_waypoint_altitude +
(command_pitch);
if(next_waypoint_altitude>GPS_Altitude()+10000){next_waypoint_altitude=GPS_Altitude()+10000;}
if(next_waypoint_altitude<GPS_Altitude()-10000){next_waypoint_altitude=GPS_Altitude()-10000;}
}
            if(elevatorcontrolmode==4)
                {


next_waypoint_altitude=constrain(map(next_waypoint_distance,last_next_waypoint_distance,0,las
t_waypoint_altitude,target_waypoint_altitude),min(last_waypoint_altitude,target_waypoint_alti
tude),max(last_waypoint_altitude,target_waypoint_altitude)); //önceki ve sonraki waypoiny
yüksekliklerinin interpolasyonu
                }
            altitudeerror=next_waypoint_altitude-GPS_Altitude();
            altitudeerroraccumulator=saturatef(altitudeerroraccumulator+altitudeerror/1000,-
Ki_altitude_limit,Ki_altitude_limit);
            command_pitch=
saturatef(Kp_altitude*altitudeerror/1000+Ki_altitude*altitudeerroraccumulator,Lo_saturate_alt
itude,Hi_saturate_altitude);
         }
```

```
//GPS_Read();

     if ((aeleroncontrolmode!=2)&&(aeleroncontrolmode!=0)){

if(aeleroncontrolmode==3){next_waypoint_direction=next_waypoint_direction+(command_roll)*0.1;
next_waypoint_direction=correctangle(next_waypoint_direction);}
             if(aeleroncontrolmode==5){next_waypoint_direction=home_direction;}
             if(aeleroncontrolmode==4)
                 {
                 next_waypoint_longitude=wplon[next_waypoint_id];
                 next_waypoint_latitude=wplat[next_waypoint_id];
                 target_waypoint_altitude=wpalt[next_waypoint_id];

                                          next_waypoint_direction=constrain(
Kp_crosstrack*float(crosstrackerror),-
crosstrackcorrectionlimit*100,crosstrackcorrectionlimit*100)+(
CalcBearing(float(GPS_Lattitude())/10000000,float(GPS_Longitude())/10000000,float(next_waypoi
nt_latitude)/10000000,float(next_waypoint_longitude)/10000000));

next_waypoint_distance=CalcDistance(float(GPS_Lattitude())/10000000,float(GPS_Longitude())/10
000000,float(next_waypoint_latitude)/10000000,float(next_waypoint_longitude)/10000000);
                                        if(next_waypoint_distance<100000){
                                         crosstrackerror=-
sin(dtor(float(correctangle(trackbearing)-
correctangle(next_waypoint_direction))/100))*float(next_waypoint_distance);
                                           }
                 if(next_waypoint_distance<=waypoint_accept_distance) //waypoint e ulaşılınca
yapılacaklar listesi
                   {
                   ++next_waypoint_id;
                   next_waypoint_id=next_waypoint_id %wpcount;
                   rollerroraccumulator=0;


trackbearing=CalcBearing(float(next_waypoint_latitude)/10000000,float(next_waypoint_longitude
)/10000000,float(wplat[next_waypoint_id])/10000000,float(wplon[next_waypoint_id])/10000000);
//erişilen ve bir sonraki waypoint arasındaki çizginin kuzey çizgisi ile yaptığı açı

                   next_waypoint_longitude=wplon[next_waypoint_id];
                   next_waypoint_latitude=wplat[next_waypoint_id];

                   last_waypoint_altitude=target_waypoint_altitude;  //burası önemli sıradaki
waypointe giderken o anki yükseklikten sıradakine gitmek yanlıştır. O an olması gereken
yükseklikten sıradakine gitmek lazımdır.!!
                                                                     //hedeflenen yükseklik
yürüngesinin takibidir
                   target_waypoint_altitude=wpalt[next_waypoint_id];

last_next_waypoint_distance=CalcDistance(float(GPS_Lattitude())/10000000,float(GPS_Longitude(
))/10000000,float(next_waypoint_latitude)/10000000,float(next_waypoint_longitude)/10000000);

                   }
                 }

             // magnetic sensörsüz düzeltme kodu. imu v3 te magnetometre var
headingerror=correctangle(next_waypoint_direction-
(GPS_Ground_Course()+headingcorrection))*0.01;


             next_waypoint_direction=correctangle(next_waypoint_direction);
             headingerror=correctangle(next_waypoint_direction-(ang_z*100))*0.01;  //gps den
acilar 100 ile çarpılmış geliyor

             headingerroraccumulator=saturatef(headingerroraccumulator+headingerror/1000,-
Ki_heading_limit,Ki_heading_limit);
             command_roll=
saturatef(Kp_heading*headingerror+Ki_heading*headingerroraccumulator,Lo_saturate_heading,Hi_s
aturate_heading);
          }

     if (throtlecontrolmode==1){
           targetairspeed=command_thro*100;
           airspeederror=(float(targetairspeed)/100)-airspeedf;
```

```
          //airspeederror=airspeedf;
        }


     rollerror=command_roll-roll;
     pitcherror=command_pitch-pitch;

     rollerrordot=-ang_xdot;//(-ang_xdot+oldrollerrordot*8)/9;
     pitcherrordot=-ang_ydot;//(-ang_ydot+oldpitcherrordot*8)/9;
     yawerrordot=ang_zdot;//(ang_zdot+oldyawerrordot*8)/9;

     rollerrordotdot=-ang_xdotdot;
     pitcherrordotdot=-ang_ydotdot;

     rollerroraccumulator=rollerroraccumulator+rollerror*(mdeltat*0.0000001);
     rollerroraccumulator=constrain(rollerroraccumulator,-Ki_roll_limit,Ki_roll_limit);

     pitcherroraccumulator=pitcherroraccumulator+pitcherror*(mdeltat*0.0000001);
     pitcherroraccumulator=constrain(pitcherroraccumulator,-Ki_pitch_limit,Ki_pitch_limit);


control_aile=int(mapf(saturatef(staticpressurecorrection()*(Kff_roll*command_roll+Kp_roll*rol
lerror+Kd_roll*rollerrordot+Kdd_roll*rollerrordotdot+Ki_roll*rollerroraccumulator),Lo_saturat
e_roll,Hi_saturate_roll),-45,45,-500,500));

control_elev=int(mapf(saturatef(staticpressurecorrection()*(Kff_pitch*command_pitch+Kp_pitch*
pitcherror+Kd_pitch*pitcherrordot+Kdd_pitch*pitcherrordotdot+Ki_pitch*pitcherroraccumulator),
Lo_saturate_pitch,Hi_saturate_pitch),-45,45,-500,500));
      control_rudd=int(control_rudd_sign*command_rudd*10+cmdoff4);
      control_thro=int(cmdoff3+saturatef( Kff_airspeed*command_thro +
Kp_airspeed*airspeederror,Lo_saturate_airspeed,Hi_saturate_airspeed)*10); //kontrol hesabı 0
ile 100 arası değer alır 10 ile çarpıp cmdoff ile toplayıp pwm olusturuyoruz

 //GPS_Read();
      if (elevatorcontrolmode==0){control_elev=int(command_pitch*10);}  //kontrol yok direkt
input device secilmiş
      if (aeleroncontrolmode==0){control_aile=int(command_roll*10);}  //kontrol yok direkt
input devie secilmiş
      if (throtlecontrolmode==0){control_thro=int(command_thro*10+cmdoff3);}  //kontrol yok
direkt input devie secilmiş
      }
      else
      {
      control_aile=int(10*(command_roll));   //burdaki kod kontrol mode0 olan durumdada aynı
olmalı yukarı kopyalamayı unutma !!
      control_elev=int(10*(command_pitch));    //burdaki kod kontrol mode0 olan durumdada
aynı olmalı yukarı kopyalamayı unutma !!
      control_rudd=int(control_rudd_sign*command_rudd*10+cmdoff4);
      control_thro=int(getmidoldpos(2));
      }

     if(elevonmixer!=1){
     servopwm(1,(1500+control_roll_sign*(control_aile+(cmdoff1/10))));   //steady flight
çalışıyor
     servopwm(2,(1500+control_pitch_sign*(control_elev+(cmdoff2/10)))); //steady flight
çalışıyor
     }else
     {
     int tcr=control_roll_sign*(control_aile+(cmdoff1/10));
     int tcp=control_pitch_sign*(control_elev+(cmdoff2/10));
     servopwm(1,(1500-tcr-tcp));   //steady flight çalışıyor
     servopwm(2,(1500-tcr+tcp)); //steady flight çalışıyor
     }

    /* control_aile=(1500+control_roll_sign*(control_aile+(cmdoff1/10)));
     control_elev=(1500+control_pitch_sign*(control_elev+(cmdoff2/10)));
     servopwm(1,control_aile);
     servopwm(2,control_elev);*/

     servopwm(3,(control_thro));
     servopwm(4,(control_rudd));
     servopwm(5,(getmidoldpos(5)));  //flap kanalı
     servopwm(10,(getmidoldpos(5)));
     servopwm(8,(getmidoldpos(4)));
```

```
     //camera_pan=map(int(cam1),0,255,-90,90);
     //camera_tilt=map(int(cam3),0,255,-90,90);
     //camera_roll=map(int(cam2),0,255,-90,90);


     float incspeed=0.1;
     int povmsg=camerapov % 100;  //pov datası nötr pozisyonda 65535 veriyor. msg yokken mod
100 35 veriyor.
     if(povmsg>=35){povmsg-=35;}  //bunu düzeltmek için bu işlem yapılıyor. 35 ayrı mesaj
verilebilir veya 5bit ayrı binary kod

     if(povmsg==1){camgyro=1;}
     if(povmsg==2){camgyro=0;}

     //Serial.println(povmsg,DEC);


     switch(long(camerapov/100)){
     case 0:
     camera_tilt_inc=-incspeed;
     camera_pan_inc=0;
     //camera_tilt=50;

     break;
     case 90:
     camera_tilt_inc=0;
     camera_pan_inc=incspeed;
     break;
     case 180:
     camera_tilt_inc=incspeed;
     camera_pan_inc=0;
     break;
     case 270:
     camera_tilt_inc=0;
     camera_pan_inc=-incspeed;
     break;

     case 99:
     camera_tilt=0;
     camera_pan=0;
     camera_tilt_inc=0;
     camera_pan_inc=0;
     break;

     default:
     camera_tilt_inc=0;
     camera_pan_inc=0;
     break;
     }

     if(millis()-camerapovmillis>1000){
     camera_tilt_inc=0;
     camera_pan_inc=0;
     }

     camera_pan=saturatef(camera_pan+camera_pan_inc,-55,55);
     camera_tilt=saturatef(camera_tilt+camera_tilt_inc,-55,55);

     //Serial.print("camera pan ");Serial.println(camera_pan,DEC);
     float aang_x=ang_x+Kd_cameraroll*ang_xdot;
     float aang_y=ang_y+Kd_cameratilt*ang_ydot;

     control_camz=camera_pan;
     control_camy=camera_tilt-aang_y*cos(dtor(camera_pan))+aang_x*sin(dtor(camera_pan));
     control_camx=camera_roll-aang_x*cos(dtor(camera_pan))-
aang_y*sin(dtor(camera_pan));//*cos(dtor(control_camy));

     //servopwm(6, map(camera_pan_trim,0,255,1000,2000)+int(control_camz*ppd_camz));    //pan
cam
     //servopwm(7,map(camera_roll_trim,0,255,1000,2000)+int(control_camx*ppd_camx));
//roll cam
     //servopwm(9, map(camera_tilt_trim,0,255,1000,2000)+int(control_camy*ppd_camy));
//tilt cam
```

```
     if(camgyro==0){
//     servopwm(6, -map(cam1,127,255,0,500)+map(camera_pan_trim,0,255,1000,2000));    //pan
cam
//     servopwm(7,-map(cam2,127,255,0,500)+map(camera_roll_trim,0,255,1000,2000));    //roll
cam
//     servopwm(9, -map(cam3,127,255,0,500)+map(camera_tilt_trim,0,255,1000,2000));    //tilt
cam

     servopwm(6, -map(camera_pan,0,45,0,500)+map(camera_pan_trim,0,255,1000,2000));    //pan
cam
     servopwm(7,0+map(camera_roll_trim,0,255,1000,2000));    //roll cam
     servopwm(9, -map(camera_tilt,0,45,0,500)+map(camera_tilt_trim,0,255,1000,2000));
//tilt cam

     }else{
     servopwm(6, int(control_camz*ppd_camz)+map(camera_pan_trim,0,255,1000,2000));    //pan
cam
     servopwm(7,int(control_camx*ppd_camx)+map(camera_roll_trim,0,255,1000,2000));    //roll
cam
     servopwm(9, int(control_camy*ppd_camy)+map(camera_tilt_trim,0,255,1000,2000));    //tilt
cam
     }

if(controlcounter%30==0){
distance_from_home=CalcDistance(float(home_latitude)/10000000,float(home_longitude)/10000000,
float(GPS_Lattitude())/10000000,float(GPS_Longitude())/10000000); }

    //*****************************control code end*****************************
    //**********************************************************************
    timer_old = timer;
    mtimer_old = mtimer;
    oldswitch_on=switch_on;

    rawairspeeddata=(80*rawairspeeddata+analogRead(2))/81;
    float inroot=( 2000.0*(rawairspeeddata-512.0)*2.0);
    if(inroot>0){airspeedf=(3.6*1.2*sqrt(inroot)*0.04);}else{airspeedf=0;}
    airspeedf=airspeedf-float(airspeedoffset);

    airspeed=int(airspeedf);
   bitWrite(statuscode,5,(airspeedf<40)&&(airspeedf>15));
   bitWrite(statuscode,6,(airspeedf>80));
}
```

CURRICULUM VITAE

Name Surname: Selman TOSUNOGLU

Place and Date of Birth: İSTANBUL 12.06.1979

Address: Cumhuriyet Mah. Şehit Ali Atik Cad. No 15/7 Şekerpınar Çayırova Kocaeli

E-Mail: selmantosunoglu@gmail.com

B.Sc.:Astronautical Engineer

Professional Experience and Rewards:

2003 -2009 Mito A.Ş. : Technical Manager

2010 – present Altınay Robotics : Project Manager