



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Megbízható kommunikációs kapcsolattal rendelkező földi irányító állomás fejlesztése UAV-hoz

SZAKDOLGOZAT

Készítette

BÖJTI PASZKÁL

Konzulensek

VÖRÖS ANDRÁS, DR. BARTHA TAMÁS

2013. december 12.

Tartalomjegyzék

Kivonat	4
Abstract	5
1. Bevezetés	6
1.1. Motiváció	7
1.2. Előzmények	7
1.2.1. Az MTA SZTAKI ORCA 2 felépítése	8
1.3. Földi irányító állomás	12
1.3.1. Mi a földi irányító állomás feladata?	12
1.3.2. Hibatűrő kommunikáció kialakítása	13
1.3.3. Grafikus felhasználói felület	13
1.4. Földi irányító állomások bemutatása és összehasonlítása	13
1.4.1. ArduPilot	14
1.4.2. Paparazzi	16
1.4.3. MicroPilot Horizon	17
1.4.4. OpenPilot	19
1.4.5. QGroundControl	20
1.4.6. HappyKillmore	20
1.4.7. Összehasonlítás	21
2. Tervezés	23
2.1. Kommunikáció megvalósítása	23
2.2. Adatok fogadása	23
2.2.1. Protokoll	23
2.3. Adatok küldése	25
2.3.1. Protokoll	25
2.4. Grafikus felület	26
2.4.1. Főképernyő	27
2.4.2. Tervezés képernyő	27
2.4.3. Diagnosztikai képernyő	27
2.4.4. Terminál képernyő	27
2.5. Használt technológiák bemutatása	28
2.5.1. .NET	28

2.5.2. GMap.NET	29
3. Megvalósítás	30
3.1. Program felépítése	30
3.1.1. Kapcsolódás megvalósítása	31
3.1.2. Redundáns adatok feldolgozása	31
3.1.3. Redundáns adatok hibadetektációja	31
3.2. Megjelenítési felületek	32
3.2.1. Főképernyő	32
3.2.2. Tervezés képernyő	34
3.2.3. Diagnosztikai képernyő	35
3.2.4. Terminál képernyő	36
3.3. Fejlesztés menete	37
4. Összefoglalás	39
Függelék	42
F.1. Program elindítása	42
F.2. Térkép letöltése	42

HALLGATÓI NYILATKOZAT

Alulírott *Böjti Paszkál*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (Böjti Paszkál, Megbízható kommunikációs kapcsolattal rendelkező földi irányító állomás fejlesztése UAV-hoz, angol és magyar nyelvű tartalmi kivonat, 2013, Vörös András, Dr. Bartha Tamás) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hállózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedélyteljes titkosított diplomatervezet esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2013. december 12.

Böjti Paszkál
hallgató

Kivonat

Napjainkban egyre nagyobb teret hódít a pilóta nélküli légi járművek alkalmazása. Az 1960-as években a hadszíntéren jelentek meg először, ahol megfigyelésre, felderítésre és olyan feladatokra használták, ahol kockázatos lett volna emberi életet veszélyeztetni. Az utóbbi években praktikussága, alacsony üzemeltetési költségei miatt más területeken is hasznosnak bizonyult ez a technológia, pl. geológia mintázatok kutatása, mely az emberi perspektívából nehezen észlelhető, tűzoltósági alakulatok koordinálása, otthoni hobby felhasználás.

Az MTA-SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumában kidolgozott szabályozó algoritmusok gyakorlatba való átültetésére egy pilóta nélküli járművet hoztak létre, mely a biztonságos üzemeltetés mellett, illetve az esetlegesen előfordulható hibák ellen redundáns hardver elemekkel védekezik. Szakdolgozatom keretében ezen repülő földi állomásához használható programot dolgoztam ki, mely a redundánsan küldött rádiójelek feldolgozására és megfelelő megjelenítésre használandó. A szoftver használatával a földi személyzet mozgó térkép alapú vizualizációt láthatja az aktív útvonalpontokat és a gép útvonalát, míg a diagnosztikai adatokat és esetleges hibákat egy másik nézetben áttekintheti. Mindezek mellett lehetőség nyílik repülési terv meghatározására és feltöltésére, aminek fordulópontjait a repülőgép követni fogja.

Abstract

Nowadays the usage of unmanned aircrafts shows an increasing number. UAVs appeared for the first time in the 1960s in military use, where it would have been risky to endanger human life, for example observation and discovery tasks.

In recent years, because of its low running costs and practical usage, this technology has proved that it is useful in other areas , eg. geology research, helping fire units and hobby use.

MTA-SZTAKI Rendszer és Irányításelméleti Kutatólaboratórium has developed control algorithms, which was elaborated into practice by an unmanned vehicle. In addition to the safe operation and for potentially active faults occur, it is defended with redundant hardware elements. In the context of this thesis I worked out a software for the ground station of this airplane, which is used to process and display the redundantly sent radio signals. The ground staff can see a map-based visualization of the active waypoints and routes the machine, diagnostic data. It is possible to create and upload flight plan, which contains the turning points which the aircraft needs to follow.

1. fejezet

Bevezetés

A pilóta nélküli légijármű, azaz UAV (Unamanned Aerial Vehicle) gondolata egészen a XX. század elejére nyúlik vissza, amikor az I. világháborúban egy olyan távirányítású repülőt alkottak, amely robbanószerrel a fedélzetén a célpontba csapódva okozott kárt. Később a vietnámi háborúban az UAV-k több mint 3000 küldetésben vettek részt. Akkoriban a technológiai korlátok miatt az alkalmazott UAV-k fő funkcionálisája videófelvétel készítése volt. Az akkori eszközök egy meghatározott útvonalat tudtak bemenetben (ami tipikusan egyenes szakaszokkal leírt pálya volt, egyes pontokon a megfigyelés érdekében körökkel kiegészítve) repülve, majd a bevetés végén a bázisra való visszaérkezés volt az utolsó feladatauk.

Idővel a technika fejlődésének köszönhetően az UAV-k egyre összetettebb feladatak elvégzésére lettek képesek. A fejlettek rádiótechnika által biztosított nagyobb átviteli sebességek köszönhetően lehetővé vált, hogy a gépet irányító operátor valós időben, monitoron keresztül kezelhesse a távirányítású légijárművet. A korszerű UAV-k ezért több üzemmódot is támogatnak, amelyek között megtalálható az említett távirányítás, valamint a fedélzeti intelligenciára támaszkodó önálló repülés, azaz az autonóm működés is. Az alkalmazási területek köre is szélesebb lett, így a katonai feladatak mellett a polgári repülés, a mezőgazdaság és a katasztrófaelhárítás is rendszeresen alkalmaz ma már pilóta nélküli légijárműveket. Az autonóm működés emellett a pilóta által vezetett kaptonai és polgári repülőgépekben is egyre szélesebb felhasználási teret nyert. Ennek oka, hogy fontos és célszerű is az emberi terhelés csökkentése, ezzel növelte a repülésbiztonságot és csökkentve a költségeket. Utasszállító gépek esetében például a robotpilóta elvégez minden olyan korrekciót, amelyhez azelőtt a pilóta folyamatos figyelme, koncentrációja volt szükséges. Ám hiába a fejlett eszközkháttér, mind a pilótával, mind a pilóta nélküli repülő légijárművek teljes körű automatikus üzemeltetése egyelőre távoli cél. Ma még az emberi beavatkozásnak rendelkezésre kell állnia olyan helyzetekben, amelyekre nincs előre felkészítve az az automatikus irányítórendszer. UAV-k esetében ilyen beavatkozásokhoz létfontosságú, hogy az operátor lássa a gép aktuális pozíóját, diagnosztikai adatait. Ezt a feladatot látja el az ún. földi irányító állomás, azaz GCS (Ground Control Station). Szakdolgozatom célja egy meglevő pilóta nélküli légijármű földi irányító állomásának megtervezése és elkészítése, a megbízható működés követelményeinek figyelembe vételevel.

1.1. Motiváció

Az MTA SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumában folyó kutatások eredményének demonstrálásához szükség volt egy kézzelfogható eszköz megalkotására. A kifejlesztett szabályozó algoritmusok működésének bemutatásának az egyik látványos módja egy autonóm repülőgép megépítése. Az autonóm légijármű stabil repülési tulajdonságait garantáló algoritmusok kidolgozásához és implementálásához elengedhetetlen a kormányszervek harmonikus mozgatása és a tolóerő szabályozása. Abban az esetben ha a repülőgépet feladatakkal látjuk el, pl. fordulópontokat követve térképezze fel az alatta lévő területet, már számolni kell a széllel, ami eltérítheti az útvonaláról. Fontos, hogy az ehhez hasonló környezeti hatásokra a gép megfelelően reagáljon.

Mivel egy teljesen felszerelt repülő összeállítása, felprogramozása nagy szakértelmet, sok időt, energiát és nem utolsó sorban anyagi ráfordítást igényel, egy esetleges meghibásodás jelentős kárt okozna. Ezen okok miatt felmerült az igény a repülő megbízhatóságának növelésére, így a most folyamatban lévő „Nagy megbízhatóságú pilóta nélküli légijármű projekt” keretében egy olyan avionikai rendszer fejlesztése is folyik, amelynek célja minden egyes repülőgép alrendszer meghibásodásának diagnosztizálása és hiba esetén a diagnosztikai információkat felhasználva a repülőgép átkonfigurálása.

Feladatom egy olyan – grafikus felhasználói felülettel ellátott – földi irányító állomás kifejlesztése egy biztonságkritikus robotrepülőhöz, amely képes redundáns kommunikációs csatornán küldött adatok kezelésére és megjelenítésére. Ehhez több részfeladat megoldása is szükséges. Első lépésként meg kell oldanom a kapcsolatot biztosító modem jeleinek vételét. Mivel a robotrepülőgép hibatűrő kialakítása révén ez az egység is redundánsan szerelt, így az adatok két független csatornán, párhuzamosan érkeznek a földi irányító állomáshoz. Mivel az így beérkező adatok szükségszerűen eltérnek egymástól, ennek az eltérésnek a kijelzése és kezelése is megvalósítandó. Továbbá, mivel a földi irányító állomáshoz a repülőt távolról megfigyelő és szükség szerint irányító földi személyzet kiszolgálására készül, így a legfontosabb cél, hogy ők a repülővel kapcsolatos információkat könnyen értelmezhetsek, és az esetleges meghibásodásokról is időben értesüljenek.

1.2. Előzmények

A jelenlegi repülő egy az MTA SZTAKI munkatársai által épített 3.2 m fesztávolságú modell, amelybe diagnosztikai és hibadetekciós célokra egyedi tervezésű komponenseket szereltek be, mivel a kereskedelmi forgalomban beszerezhető szervő motorok nem szolgálnak elég információval a kitérésükiről, fogyasztásukról, gondoskodni kellett ezen adatok mérhetőségéről.

1.2.1. Az MTA SZTAKI ORCA 2 felépítése



1.1. ábra. MTA SZTAKI ORCA 2

Architektúra

A MTA SZTAKI „Nagy megbízhatóságú pilóta nélküli légijármű” projektjében elsődleges szempont, hogy egy komponens meghibásodása ne okozza a gép vesztét, tehát a rendszerben ne maradjon SPOF (Single Point Of Failure)¹. Ezt redundanciával érhetjük el, azaz a repülőgépen duplikáljuk azokat az elemeket, melyek nélkülözhetetlenek a levegőben maradáshoz:

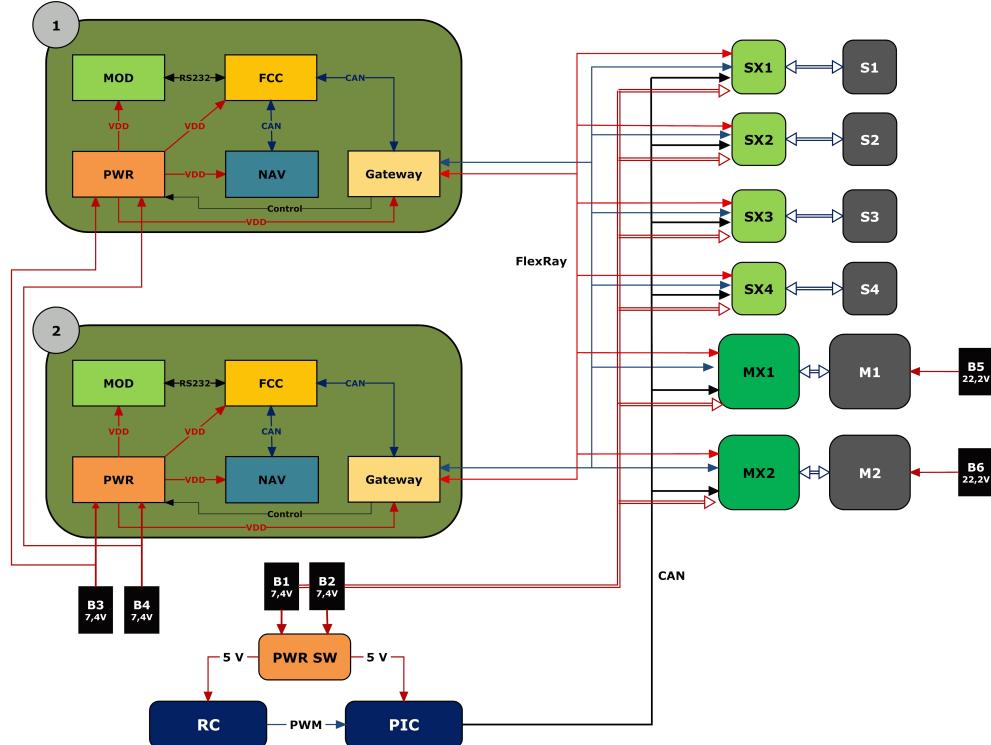
- motor
- kormányfelületek és ezeket mozgató motorok
- tápellátás
- központi számítógép

A 1.2. ábrán látható a kialakított architektúra. Elemei:

- **FCC** (Flight Control Computer) repülőgép irányítása
- **NAV** (Navigation) GPS, nyomásmérő, orientációmérő
- **MOD** (Modem) modem, kommunikáció
- **Gateway** CAN-FlexRay átjáró
- **PWR** (Power) feszültségválasztó
- **M1, M2** (Motor) motorok
- **S1, S2, S3, S4** (Servo) szervók

¹olyan meghibásodás, mely ha bekövetkezik, az egész rendszer hibás működéséhez vezet

- **Sx, Mx** (Microcontroller) szabályzóelektronika
- **RC** (Remote Control) távirányító, manuális vezérlés
- **PIC** (PIC Microcontroller) PWM²-CAN³ átalakítás



1.2. ábra

A központi számítógépek(ábrán zölddel jelölve 1-es 2-es) 1–1 szendvics panelen helyezkednek el, központi elemük az FCC, mely az irányításért felelős. A navigációs eszköz szolgáltatja a GPS-ből érkező magasság és pozíció adatokat, az IMU (Inertial Measurement Unit)⁴-ból érkező orientációt, a nyomásmérőből a légsebesség és magasság értékeit. Ezen egy mikrokontroller előfeldolgozást végez, így már csak a ténylegesen feldolgozandó információval kell az FCC-nek számolnia. Az FCC és a NAV közötti kommunikáció CAN(Controller Area Network) buszon keresztül zajlik. Az ábrán látható S és M-mel jelölt elemek rendre a szervó motorok és a meghajtásért felelős motorok, melyek redundáns FlexRay kommunikációs csatornán kapják az utasításokat az FCC-ből.

A CAN–FlexRay és FlexRay–CAN átalakítást a Gateway egység végzi. A szervók és motorok nem szolgálnak elég információval saját állapotukról, így ezeket átalakították, hogy a FlexRay hálózatra illesztésért felelős szabályozóelektronika (Sx, Mx) megfelelően működhessen. Ezekre bármilyen szabályzóalgoritmus írható, hibadiagnosztikai célokra fa-

²Pulse-Width Modulation, impulzusszélesség moduláció, mely a szabályzást nem feszültségszint növelésével, hanem annak időtartamával éri el

³Control Area Network, egy számítógépes hálózati protokoll és adatbusz szabvány melyet mikrokontrollerek és egyéb gazdaszámítógép nélkül működő eszközök kommunikációjára terveztek [5]

⁴orientáció- és gyorsulásmérő berendezés

ult detection filtert vagy Kármán szűrő alkalmazható. A szervók mágneses enkódere a kitérésről, a motorok elektronikája a fogyasztásról ad információt.

Hibatírásból adódóan az energiaforrások is redundánsan szerepelnek, a központi számítógépek a B3 és B4-gyel jelölt akkumulátorból nyerhetnek energiát, a választást a PWR néven jelzett egység végzi. Különböző stratégiák választhatók az akkumulátor átkapcsolását illetően, lehetséges mindenkor a legnagyobb feszültséggel operálót választani vagy egyiket lemeríteni bizonyos százalékig és ezután váltani. A B5 és B6 a motorokat hivatottak ki-szolgálni, ezek a legnagyobb fogyasztásúak, így ezek kapacitása a legnagyobb. B1 és B2 biztosítja az aktuátoroknak, az hozzájuk tartozó vezérlőknek az áramforrást.

Mivel a repülőgép jelenleg csak távirányítással tud felszállni, így a távvezérlő egység (RC) és a PWM-CAN átalakításért felelős PIC is ezt a két akkumulátort használja. A PIC közvetlenül a szabályzó elektronikához csatlakozik CAN buszon, mivel jelen felépítésben ez a legközvetlenebb módja a kézi irányításnak.

Dolgozatom szempontjából a legérdekesebb egység a modem mely az FCC-vel soros porton kommunikál.

Vezeték nélküli kommunikáció

A fedélzeti MOD egységek XBee-PRO 868 típusú modemek [6] (lásd 1.3. ábra), melyek alacsony fogyasztásuk és nagy hatótávolságuk miatt ideálisak egy szűkös energiaforrással rendelkező robotrepülőbe. Ugyanilyen modemek csatlakoznak a földi irányító egységhez, melyek a fedélzeti modemet hasonlóan soros porton keresztül küldik a vett jeleket. A modempárok közötti vezeték nélküli protokoll: 802.15.4, amivel a 1.2.1 fejezet foglalkozik.

XBee-PRO 868 modem

A kiválasztott modem kétféleképpen képes kommunikálni a csatlakoztatott eszközzel:

- AT transzparens
- API (Application Programming Interface)⁵ csomagküldés

AT: Alapértelmezetten transzparens módon zajlik a kommunikáció a modemhez csatlakoztatott eszköz és a modem között [7]. Ennek lényege, hogy a csatlakoztatott eszköz számára a kommunikáció ugyanúgy zajlik, mintha közvetlen összeköttetésben állna a másik eszközzel. A modem a soros portján bejövő UART (Universal Asynchronous Receiver/Transmitter) üzeneteket rádiójelekké alakítja, melyet a virtuális kapcsolat végpontja fogad és visszaalakít. Egy pont-pont kapcsolat kiépítése ebben a módban a legegyszerűbb.

API: API mód lényege, hogy a csatlakoztatott eszköz a modem által biztosított API-n keresztül kommunikál. Az AT módhoz képest nagyobb rugalmasságot biztosít, mivel egy csomag a programozó szándéka szerint tartalmazhatja a küldő és fogadó eszköz címét, a csomag típusát, checksum-ját. A küldő fél a csomagok megérkezéséről ACK (Acknowledgement)⁶ jelzéssel bizonyosodhat meg, ennek elmaradásnál újból küldés lehetséges. A

⁵előre megírt komponensek használatához biztosít interfész

⁶nyugtázó üzenet, melyet a fogadó küld a feladónak egy csomag sikeres vétele esetén

csomagok többletinformációjával lehetséges broadcast üzenetek⁷ küldése, egy-több kapcsolt felépítése.



1.3. ábra. XBee-PRO 868

IEEE 802.15.4 vezeték nélküli protokoll

Az IEEE (Institute of Electrical and Electronics Engineers)-nek egy szabványa a 802.15 mely a WPAN (Wireless Personal Area Network) hálózatokkal foglalkozik, hét alcsortörő közül a 802.15.4 [9] a Low-Rate WPAN nevet viseli. Ez a szabvány a kis fogyasztású, olcsó és alacsony sávszélességű vezeték nélküli kommunikációt foglalja magában. Az OSI (Open Systems Interconnection)⁸ modell első (fizikai) és második (adatkapcsolati) rétegét valósítja meg (1.4. ábra), erre építkezik a ZigBee [8] cég által specifikált protokoll verem (1.5. ábra), ami a hálózati és az alkalmazási réteggel egészíti ki.

A fizikai réteg átjárást biztosít a felette lévő adatkapcsolati rétegnek és összekötetést teremt a hardverrel, specifikálja a működési feszültséget, szabályozza a használandó frekvenciát, ami Európában 868-868.6 MHz, Észak-Amerikában 902-928 MHz, világszerte 2.4-2.4835 GHz-es ISM (Industrial, Scientific and Medical)⁹ tartományban helyezkedik el.

Az adatkapcsolati rétegnek a feladata hibamentes átvitel biztosítása két pont között hibajavítással, hibajelzéssel és forgalomszabályozással. A biteket keretekbe ágyazva küldi a felsőbb rétegnek. Egy keret maximális mérete 127 bájt, formátuma a IEEE 802.15.4-2011-es szabványban specifikált.

A hálózati réteg feladata az eszközök hálózathoz való le- és felcsatlakozásának kezelése, illetve szomszédos eszközök felderítése.

Az alkalmazási réteg összeköti a hálózati réteg adatkommunikációs és menedzsment részét a ZDO(ZigBee Device Objects)-k között. A ZDO TODO

A protokoll jellemzői

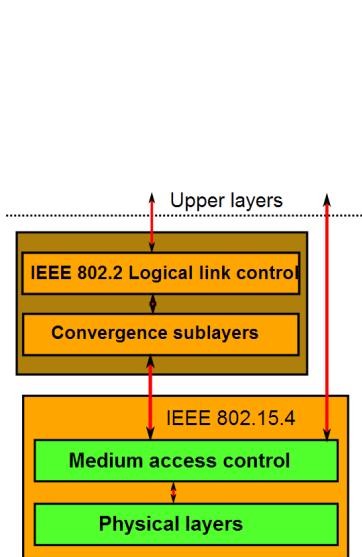
Alacsony fogyasztás: A Zigbee protokollra épülő eszközök alacsony fogyasztása abban rejlik, hogy alvó állapotból 30 ms alatt aktívra tudnak váltani, így képeses alacsony fogyasztású állapotban tartózkodni jelentős késleltetés nélkül.

Nagy hatótávolság: A 868 MHz-es verzióról a BPSK (Binary Phase-Shift Keying) [11]

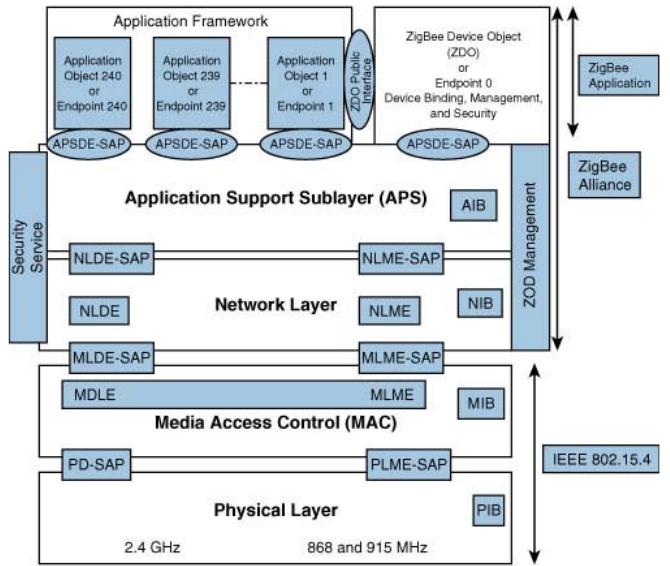
⁷olyan üzenet, melyet egy csomópont az összes vele kapcsolatban álló csomópontnak elküld

⁸rétegekbe szervezett rendszer absztrakt leírása, ami a számítógépek kommunikációjához szükséges hálózati protokollt határozza meg [10]

⁹sávok, melyek szabadon használhatóak ipari, tudományos és orvosi területen



1.4. ábra. IEEE 802.15.4 protokoll rétegei



1.5. ábra. ZigBee protokoll

és DSSS (Direct Sequence Spread Spectrum) technológiák keresztezésével érték el. A BPSK lényege, hogy a 0-1 és 1-0 átmenetet a vivőfrekvencia 180 fokos fordításával reprezentálja, így az átvitt információ jelentős zajt képes elviselni. A 802.11-es szabvány DSSS-leírása egy adatbitnek egy 11 bites kódot feleltet meg, és ezt az adatbit és egy 11 bit hosszú úgynvezett Barker-sorozat(10110111000) kizáró vagy kapcsolatával állítja elő. Egy adatbitnek tehát 11 bit felel meg, sokszoros redundanciát hordozva.

Megbízhatóság: A megbízhatóságot CSMA/CA (Carrier Sense Multiple Access Collision Avoidance) segítségével érték el. A módszer lényege, hogy mielőtt egy állomás jelet adna, megnézi, hogy a csatornán zajlik-e kommunikáció, és ha igen, akkor különböző stratégiák segítségével vár az üzenet újraküldésével.

Az XBee a Zigbee protokollt megvalósító bejegyzett márkaneve, melynek tulajdonosa a Digi cég.

1.3. Földi irányító állomás

Feladatom a földi állomás elkészítése egy program formájában, ehhez szükséges a kapcsolat kiépítése a repülőgép és a földi állomás között, melynek során meg kell oldanom a soros porti kommunikáció létrehozását majd a fogadott független adatok feldolgozását és kijelzését.

1.3.1. Mi a földi irányító állomás feladata?

Egy UAV vezetéséhez elengedhetetlen egy bázis, ahonnan a földi személyzet irányíthatja és monitorozhatja a repülést. Egy állomás általában több funkciót lát el [2]:

- Küldetés tervezése: útvonal meghatározása, illetve a célpontok kijelölése
- Küldetés végrehajtása: az operátor távirányítással hajtja végre a feladatot

- Adatok megjelenítése: megfelelő módon jelzi a repülőgép állapotát, annak esetleges hibáit

1.3.2. Hibatűrő kommunikáció kialakítása

A repülőgép és a fejlesztendő program közti megbízható kommunikációhoz az alábbi feladatok megoldása szükséges:

Modem kommunikáció

A modem soros port interface-t biztosít, egy USB-RS232 átalakítóval USB porton keresztül megoldható egy olyan számítógéppel is az összeköttetés, mely nem rendelkezik soros porttal, pl. saját energiaellátással rendelkező modern laptop. Feladataim közé tartozik, hogy biztosítsam az útvonalpontok feltöltésének lehetőségét. Ehhez kétirányú kommunikáció kiépítése szükséges.

Párhuzamos csatornák

Mivel a repülőgépen duplikáltak az elemek, így a küldő oldali modem is, a független jelek vételére megoldást kell találni és az esetleges eltérésekkel is számolni kell.

Biztonságos protokoll

Kétirányú kommunikáció kialakítása a cél, így a küldendő adatok csomagjának biztonságos protokollját ki kell dolgozni, mely az adatintegritás megőrzése érdekében hibajelzésre alkalmazható.

1.3.3. Grafikus felhasználói felület

Ahhoz hogy a kialakítandó földi irányítóállomás kezelőfelülete kényelmes és hatékonyan használható legyen, célszerű több nézeti oldal elkészítése:

- Egy áttekintő képernyő, mely a legfontosabb adatokat jeleníti meg a repülővel kapcsolatban (pozíció, sebesség, irány)
- Egy tervező modul, amin a lerepülendő útvonalhoz tartozó fordulópontok kijelölése lehetséges
- Egy diagnosztikai nézet, melyen az alacsonyabb prioritású adatok tekinthetők át
- A kommunikáció alacsony szintű megjelenítésére egy terminál ablak létrehozása, melyen a kapott nyers adatok látszódnak

1.4. Földi irányító állomások bemutatása és összehasonlítása

Számos megoldás született földi állomások GUI (Graphical User Interface)¹⁰-jainak kialakítására. Elsődleges követelmény, hogy az operátor minden a legfontosabb információkat

¹⁰grafikus megjelenítés

láthassa, ehhez szoftverergonómiaiailag kell megtervezni a grafikus felületet. Kísérleteket folytattak, milyen elrendezésben, hány képernyőn érdemes megjeleníteni az adatokat, úgy, hogy az még ne terhelje túl az operátort [3]. Ebben a kísérletben bebizonyosodott, hogy pl. egy nagy prioritású eseménynél a figyelmeztető ablak megjelenésénél érdemes hangjelzést is adni.

Alábbiakban összehasonlításra kerülnek a piacra lévő megjelenítési felületek, megoldások.

1.4.1. ArduPilot

Az Ardupilot projekt [12] létrejöttének oka, hogy otthoni körülmények között, nem ipari alkatrészekből bárki összeállíthat egy autonóm járművet, mely az előre betáplált utasításokat végrehajtja. Központi elem az Atmel cég által készített Atmel AVR mikroprocesszorra épülő Arduino platform [4]. Az Arduino egy „system on a board”, aminek lényege, hogy a felhasználó egy előre elkészített eszközt kap, amit egyből a kívánt cédra használhat fel. Az Arduino programozási nyelv segítségével az eszközzel ismerkedő programozó magas szintű utasításokkal programozhatja az eszközt, nincs szükség alacsony szintű, assembly nyelvtudásra. Erre a platformra hozták létre az ArduPilot programot, ami képes többféle autonóm járművet vezérelni:

- ArduPlane néven futó változat: robotrepülőgép
- ArduCopter: 1, 3, 4, 6, 8 propelleres helikopter
- Ardurover: 4 kerekű autó

Sikerességének fő oka a nyílt forráskód, az Arduino panel alacsony ára és a projekt mögött álló lelkes közösség. Ennek a közösségnak köszönhetően született a Mission Planner nevű szoftver, mely teljes körű támogatást nyújt a csatlakoztatott járművek útvonaltervezéséhez, grafikus megjelenítéséhez.

Főképernyő

A program funkcionalitása több nézet segítségével könnyen átlátható. A főképernyőn repülés szempontjából a legfontosabb adatok találhatóak. A repülőgép orientációját, sebességét, irányát egy műhorizonton láthatja a felhasználó, ez vizuálisan szemlélteti, hogy hány fokos bedöntéssel repül, milyen állásszöggel emelkedik. Alatta lévő területen előre beállított adatokat jelenít meg, pl. GPS magasság, GPS sebesség, szélirány (valós mágneses irány és a sebesség vektor különbségéből számítható). A felületen a legnagyobb területet a térkép foglalja el, melyen az aktuális irány, a lerepült útvonal és a fordulópontok láthatóak.



1.6. ábra

Ez a nézet akkor jöhét jól, ha olyan meghibásodás történik, melyre nincs felkészítve az irányítóegység és szükség lehet a kézi üzemmódra váltásra. Ilyen esetben előfordulhat, hogy nincs vizuális kapcsolat az operátor és a repülőgép között. Ilyen esetben csak a képernyőn látható műszerek és térkép alapján lehetséges a repülőgép irányítása.

Tervező és útvonalfeltöltő képernyő

A tervező nézet lehetőséget biztosít a lerepülendő útvonal meghatározására. Az útvonalat meghatározó útvonalpontok típusa (indulási-, forduló- vagy végpont) egy listából választható. Az útvonalpontok pozíciója egérrel módosítható, törölhető. A képernyő bal felső sarkában a kijelölt útvonaltervnek hossza látható. Ennek segítségével elkerülhető, hogy a repülés során a gép túllépje a rádiókapcsolat és a saját hatósugarát. Az elkészített tervet feltölthetjük a csatlakoztatott eszközre.

Waypoints											
WP Radius	Loiter Radius	Default Alt	<input type="checkbox"/> Absolute Alt	<input checked="" type="checkbox"/> RTL@def Alt	<input type="checkbox"/> Verify Height	Add Below					
30	45	100					Lat	Long	Alt	Delete	Up
1	TAKEOFF	0	0	0	0	10,8333060	-14,0625000	100	X		
2	WAYPOINT	0	0	0	0	4,5654736	63,2812500	100	X		
3	RETURN_TO_LAUNCH	0	0	0	0	4,5654736	63,2812500	100	X		
4	LAND	0	0	0	0	-25,4829512	3,1640625	100	X		

1.7. ábra

A program jobb felső sarkában a „Csatlakozás” gomb mindenkor látható, itt a soros port és a jelsebesség beállítása után ezzel a gombbal csatlakozik a program a kiválasztott portra.

Összegzés

A Mission Planner program felhasználói szempontból kényelmes felületet biztosít a különböző nézetekkel és a gombok átgondolt elhelyezésével. Az ArduPilot projekt egyszerűsége miatt nincs felkészítve párhuzamos csatornák kezelésére, mert az egész projekt nem használ redundanciát. Jó ötlet a „Csatlakozás” gomb minden látható elhelyezése, a repülőgép helyzetének főképernyőn, egy térképen való megjelenítése, illetve orientációjának a műhorizont segítségével történő mutatása. Hátrányai között szerepel, hogy a hibadiagnosztikai kijelzés nem megoldott.

1.4.2. Paparazzi

Az ArduPilot-hoz hasonlóan a Paparazzi projekt [13] is a könnyen, otthon összeszerelhető repülő megépítése jegyében született. Nem csak egy nyílt forráskódú robotpilótát ad, hanem egy komplett „csináld magad” készletet, amelyben a feszültségszabályzótól kezdve a GPS vevőig minden biztosított, a hozzá készült földi állomáshoz antennát, modemet és programot is rendelkezésre bocsát.

A program a következő funkciókkal rendelkezik:

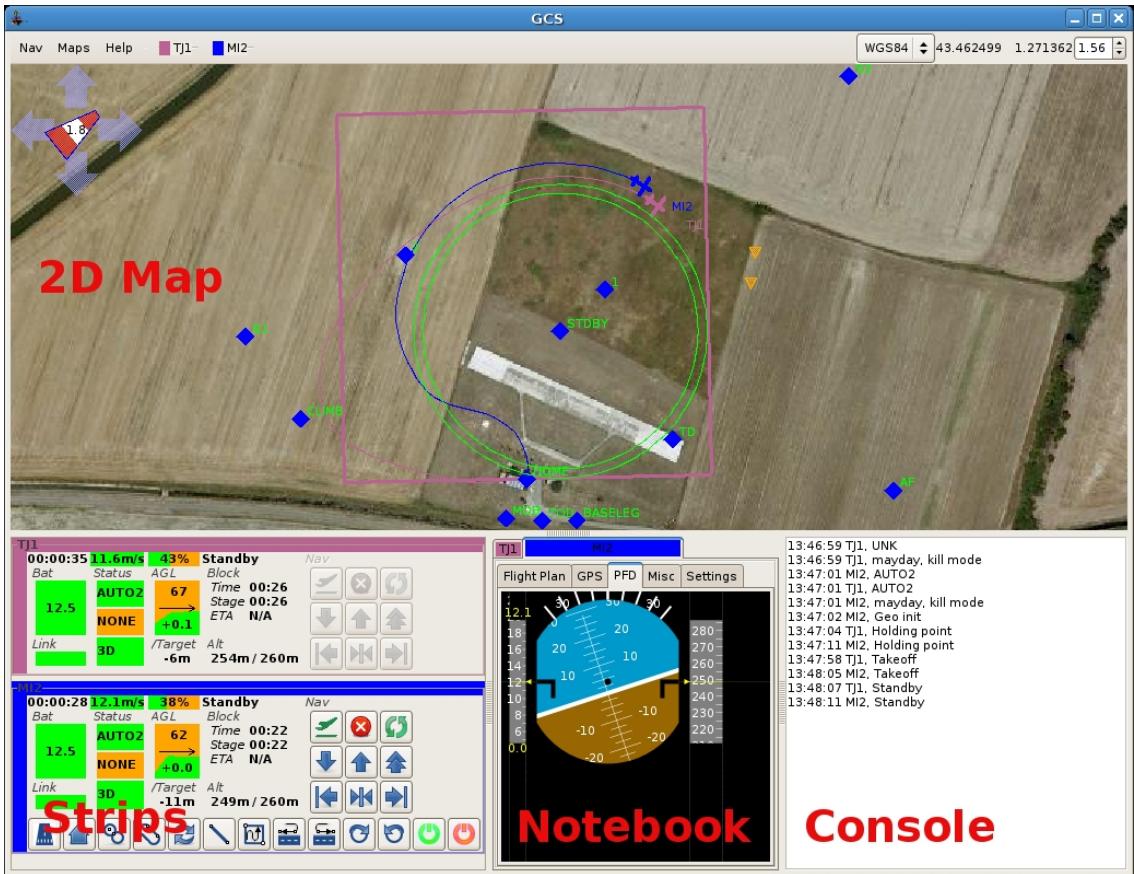
- több platform támogatása (fix- és forgószárny)
- több jármű egyidejű kezelése
- Google Maps, OpenStreetMaps, Microsoft Maps térképes megjelenítése
- küldetéstervezés
- hangjelzés

Főképernyő

Az 1.8. ábrán látható képernyő jelentős részét a térkép foglalja el, melyen a csatlakoztatott járművek lerepült útvonalra (különböző színnel jelölve) és a fordulópontok láthatóak. A járművek aktuális helyzete mellett a jármű neve, sebessége és repülési magassága is kijelzésre kerül. Bal oldalon az információs sávban mindegyik eszköz fontosabb adatai láthatóak: akkumulátor töltöttsége, sebesség, tolóerő, magasság, illetve utasítások: fel-, leszállás, megfigyelés indítása. A jobb felső sarokban a kurzor térképen lévő koordinátája látszik.

A térkép billentyűzet és egér segítségével mozgatható, nagyítható. A fordulópontok is ezen a felületen szerkeszthetők. A módosítások egy figyelmeztető üzenet jóváhagyása után töltődnek fel a repülőre, melyre az egy megerősítő válasszal reagál. Új pontot csak felszállás előtt lehetséges feltölteni.

A képernyő alsó részének közepén egy több füllel ellátott rész található, ahol kiválasztható, hogy egy műhorizont, a GPS által vett adatok, az útvonalterv vagy a beállítások látszódjanak-e.



1.8. ábra

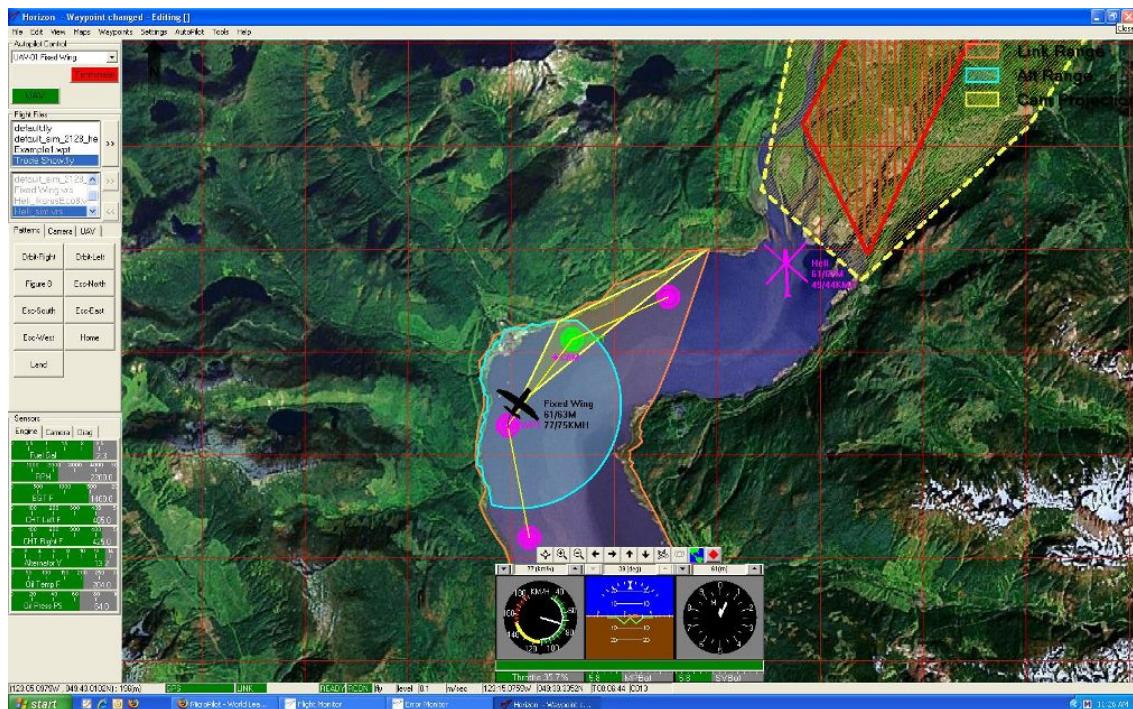
Összegzés

Ugyanaz a képernyő szolgál a navigációra és az útvonal-kijelölésre, ami szerintem nem a legjobb megoldás, mivel nincs elválasztva ez a két funkcionálitás. Egyértelműbb, ha a program a tervezéshez egy olyan perspektívát nyújt, ahol minden szükséges információ rendelkezésre áll az útvonallal kapcsolatban. Repülés közben ezek az információk nem szükségesek, sőt zavaróak is lehetnek. Összességében ez is egy nagyon jól használható program, ami minden olyan funkciót biztosít, mely egy UAV használatához szükséges.

1.4.3. MicroPilot Horizon

„Az 1995 óta működő kanadai MicroPilot [14] a világ egyik legismertebb robotpilóta gyártó cége, 65 országban több mint 750 alkalmazó használja az eszközeiket. Népszerűek az iskolákban, egyetemeken, kutató intézetekben, a gazdaság különböző területein, de a védelmi szféra is szép számmal használ robot légi járművein MicroPilot berendezéseket. Az adatátviteli csatorna ugyanazokat a funkciókat képes biztosítani, mint a programbevitelnél használt kábeles összeköttetés, így ezen keresztül repülés közben is módosítható az útvonal, a repülési paraméterek és az egyéb beállítások. A MicroPilot fedélzeti egysége ezen kívül egy rádió távirányító vevőberendezést is fogad, amely lehetőséget teremt a földi adó konzoljáról az irányítás átvételére és botkormányos kézi vezérlésre. Ezt alapvetően a fel és leszállás idejére, illetve az útvonal kritikus szakaszain használják. Amennyiben az RC

távirányítóval működő repülőgép veszti el a kapcsolatot, akkor a fedélzeti vevőberendezés Failsafe üzemmódra kapcsol és annak beállítása szerint működteti a repülőgépet. A Failsafe vagy az utolsó szervó állást őrzi meg, vagy egy előre programozott legbiztonságosabb földet érést ígérő beállításra ugrik.”[15] A hozzá készített földi irányító egység a MicroPilot Horizon nevet viseli.



1.9. ábra

Főképernyő

A repülőgépre szerelt kamera képének megjelenítése kulcsfontosságú, mivel a kezelő ezzel láthatja leginkább a repülőgép helyzetét és a megfigyelt célpontot. Ebben a módban a legfontosabb repüléssel kapcsolatos információk átlátszó háttérrel jelennek meg, így nem kell másik képernyőre tekintenie a kezelőnek. A program egyszerre több eszközhöz tud csatlakozni és azokat kiszolgálni, melyekhez külön név rendelhető. Az útvonalpontok az összes csatlakoztatott jármű között szinkronizáltak. A szakdolgozat feladata szempontjából érdekes lehet, hogy a felmerülő hibákhoz lehetőség kínálkozik különböző prioritási szintek meghatározására és hangjelzés hozzárendelésére. Így egy bizonyos szint alatti hibák nem terelik el az operátor figyelmét.

Tervező képernyő

A repülési terv könnyen, kattintással összeállítható és módosítható, repülés előtt és közben egyaránt. Az útvonal hossza folyamatosan kijelzésre kerül tervezés üzemmódban.

Összegzés

Ez a program elsősorban távirányításos vezérlés támogatására készült, aminek során az operátor irányítja a gépet. Így kulcsfontosságú a videókép átvitel támogatása és a legfelhasználóbarátabb megjelenítés. Érdekesség a POI (Point Of Interest)¹¹ gomb, mellyel egy pozíció későbbi kivizsgálásra megjelölhető, ha esetleg valami olyat lát az operátor, melyet érdemes felkeresni.

1.4.4. OpenPilot

Az OpenPilot projekt [16] célja, hogy nyílt forráskódú, magas minőségű robot pilótát kínáljon autonóm légi járművek vezérléséhez. 2009-ben alakult, azóta már több mint 200 tagja vesz részt a fejlesztésben a világ 140 országából. A jelenlegi verzió, képes irányítani fix szárnyas repülőt és helikoptert kettőtől nyolc rotorig.

A hozzá készült földi irányító program több platformon működik: Windows, Mac OS, Linux és tervben van az Android támogatása is. A program segítségével töltethető fel a panelra a fedélzeti program.

Főképernyő

A képernyő bal oldalán egy műhorizont látható, alatta a jármű orientációja külső nézetből, jobb oldalon a térkép az aktuális pozíció megjelenítésével.

Ez a nézet szabadon módosítható különféle „gadget”-ek kiválasztásával, pl. a külső nézet egy mért érték grafikus megjelenítésére kicserélhető. Ezek egy .xml fájlba elmenthetők és visszatölthetők, különböző konfiguráció igényeihez igazodva.



1.10. ábra

¹¹GPS koordinátával megjelölhető érdekes hely

Összegzés

Rengeteg beállítási lehetőséget nyújt ez a program, lelkes közösséggel áll mögötte, minden olyan fontos igényt kielégít, mely napjainkban felmerülhet egy UAV adatainak megjelenítésével kapcsolatban.

1.4.5. QGroundControl

[17] MAVLink protokollt használ a kommunikációra, a repülőgép a programon keresztül vezethető, ha átkapcsolás szükséges. A MAVLink kifejezetten GCS–UAV összeköttetésére lett kifejlesztve.

Főképernyő

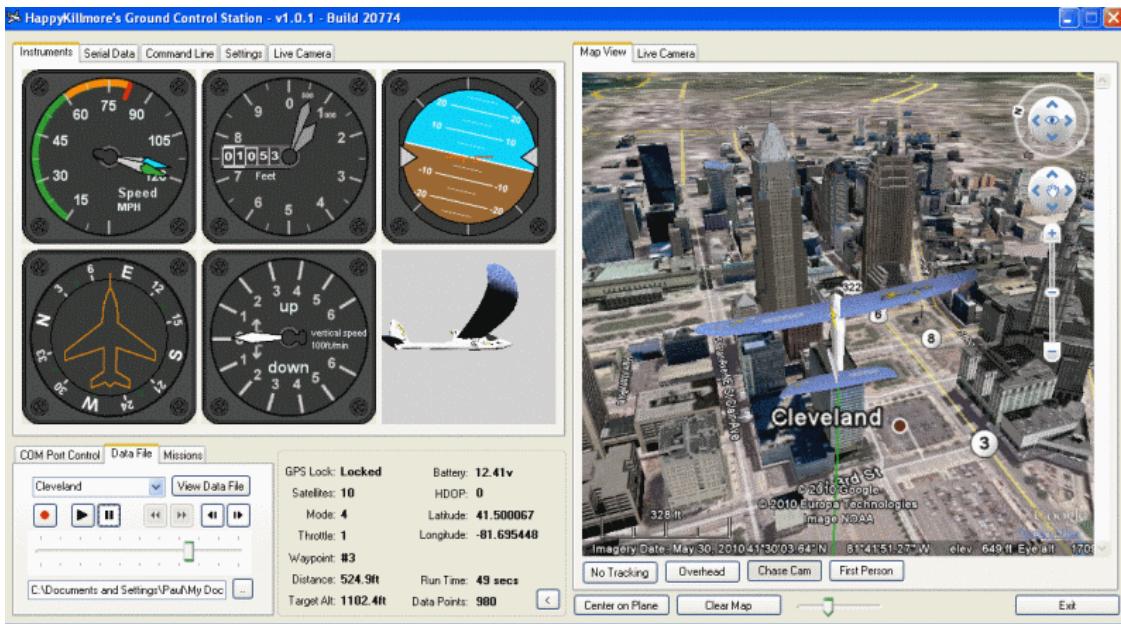
Érdekesség, hogy 3D-ben is meg tudja jeleníteni a repülő aktuális pozíóját. A protokollnak köszönhetően 255 jármű egyidejű kezelésére alkalmas. Útvonalpontok menet közben is változtathatóak.



1.11. ábra

1.4.6. HappyKillmore

[18] ArduPilot-hoz készült nyílt forráskódú GCS, a Planner-hez képest ez inkább az egyszerűség híve, kevés műszer található rajta, de ezekről minden információ megtudható. Itt is a térkép nézet dominál, oldalt mozgathatjuk számunkra megfelelő elrendezésbe a műszereket. Útvonalpontok feltöltésére szintén van lehetőség. Támogatja többek között az ArduPilot és a MAVLINK protokollokat.



1.12. ábra

1.4.7. Összehasonlítás

A felsorolt megoldások összehasonlításából kiderül, hogy egyik sem alkalmaz redundanciát a kommunikáció megvalósítására. Az összesre jellemző, hogy a repülőgép pozíciója valamilyen térképen kerül megjelenítésre. Általában a főképernyőn a sebesség, magasság adatok minden látszódnak, így az elkészítendő felületre célszerű egy térképes megjelenítést és a legfontosabb információk kijelzését megoldani. Számos program különbözőként támogatta az útvonal-kijelölés felületét a főképernyőtől, így ebből merítve készíthető egy olyan felület, melyen útvonalpontok hozzáadhatóak, módosíthatóak és feltölthetők. Mivel redundanciával nem foglalkozik egyik sem, így a dolgozat szempontjából érdekes hibadetektációra nem találunk példát. Láthattuk néhány program több eszköz párhuzamos kiszolgálását támogatta, jelenleg a feladatom egy eszközök között való csatlakozás megoldása, amely ha szükséges, látható, ezt már sikerült megoldani. A megoldások forráskódja elég változatos a választott nyelv tekintetében, mondhatni, a nyílt forráskódúak az összes manapság használatos programozási nyelvet felhasználják. A nem nyílt programokról sajnos nem találtam információt. A nyílt forráskódúak tanulmányozásával lehetőség van ötleteket meríteni a tervezéshez és megvalósításhoz.

A repülő, melyhez a programot készítem, nagy megbízhatóságú, így szükséges megoldani a két port kezelését és az azokon érkező adatok feldolgozását. Azonban

Program	Ardupilot	Paparazzi	MicroPilot	QGroundControl	HappyKillMore
Forráskód	C#, nyílt	?, nem nyílt	?, nem nyílt	C++, nyílt	VB, nyílt
Több eszköz				-	
Útvonal szerkesztése	-	+	+	-	
Előny		+	-	-	
új	-	-	-		

1.1. táblázat. Összefoglalás

2. fejezet

Tervezés

A bevezetés fejezetében ismertettem a feladatom részleteit, illetve az előző alkotásokból levont következtéseket és megoldásokat felhasználom.

2.1. Kommunikáció megvalósítása

A kommunikáció csatornánként 2 db modem segítségével történik, a modemek egymás közt vezeték nélkül csatlakoznak, felhasználói oldalon soros portot biztosítanak. Így az el-készítendő programnak elég csak a soros port jeleinek vételével foglalkoznia. A modemek adatátviteli sebessége változó lehet, így azt a felhasználó egy listából választhatja csatlakozás előtt. Átalakítóval lehetőség adatik USB-n keresztül soros port megvalósítására, így könnyen kezelhetővé válik a periféria illesztés.

2.2. Adatok fogadása

A redundanciából következően külön-külön kell kezelní a 2 párhuzamos csatornán kapott adatokat. Mivel aszinkron módon érkeznek a csomagok, így kettő FIFO tároló szükséges, mely a legutóbb küldötteket tárolja.

2.2.1. Protokoll

Az adatokat a repülőgép 2 Hz-s frekvenciával küldi, ezek az adatok csomagokban érkeznek egy előre meghatározott protokoll szerint, melyeknek a felépítése: A csomag eleje egy UUT fejlécet tartalmaz a beazonosítás végett, ezt követik az adattagok, majd az utolsó 2 bájt checksum, mely a csomag tartalmának bináris összegét tartalmazza 16 bitre csonkolva. Egy csomag fogadásánál először ezt az összeget számolom ki, ha a fogadott checksum-mal nem egyezik, akkor az egész csomag eldobásra kerül.

Telemetria csomag leírás

érték = nyers érték / skálázás - ofsztet							
bájt index	leírás	típus	bájt sorrend	változó név	offset	skálázás	mértékegység
0	start bájt 1	uint8	1/1	'U' = 85			
1	start bájt 2	uint8	1/1	'U' = 85			
2	start bájt 3	uint8	1/1	T = 84			
3	idő	uint32	1/4	get_time()	0,00	10000,00	s
4			2/4				
5			3/4				
6			4/4				
7	magasság parancs	uint16	1/2	alt_dmd	0,00	0x7FFF / 10000	m
8			2/2				
9	sebesség parancs	uint16	1/2	ias_dmd	0,00	0x7FFF / 80	m/s
10			2/2				
11	szögsebességek	uint16	1/2	smart_imu->gyr1[0]	250,00	0x7FFF / 500,0	°/s
12			2/2				
13		uint16	1/2	smart_imu->gyr1[1]	250,00	0x7FFF / 500,0	°/s
14			2/2				
15		uint16	1/2	smart_imu->gyr1[2]	250,00	0x7FFF / 500,0	°/s
16			2/2				
17	nyomás alapú magasság	uint16	1/2	tmp_P	200,00	0x7FFF / 8200	m
18			2/2				
19	IAS	uint16	1/2	smart_imu->ias	0,00	0x7FFF / 80	m/s
20			2/2				
21	Euler-szögek	uint16	1/2	ahrs->psi	180,00	0x7FFF / 360,0	°
22			2/2				
23		uint16	1/2	ahrs->theta	90,00	0x7FFF / 360,0	°
24			2/2				
25		uint16	1/2	ahrs->phi	180,00	0x7FFF / 360,0	°
26			2/2				
27	normalizált kormánykitérítések	uint16	1/2	control_cm->dr		0x7FFF	[-1..1]-re normálva
28			2/2				
29		uint16	1/2	control_cm->de		0x7FFF	[-1..1]-re normálva
30			2/2				
31		uint16	1/2	control_cm->da		0x7FFF	[-1..1]-re normálva
32			2/2				
33	normalizált gázkarállás	uint16	1/2	control_cm->dthr		0x7FFF	[0..1]-re normálva
34			2/2				
35	GPS egészség	uint16	1/2	TRUE/FALSE			
36			2/2				
37	GPS pozíció	uint32	1/4	smart_gps->POSLH.lon	90,00	0xFFFFFFFF / 180,0	°
38			2/4				
39			3/4				
40			4/4				
41		uint32	1/4	smart_gps->POSLH.lat	180,00	0xFFFFFFFF / 360,0	°
42			2/4				
43			3/4				
44			4/4				
45		uint32	1/4	smart_gps->POSLH.height	100,00	0xFFFFFFFF / 10100,0	°
46			2/4				
47			3/4				
48			4/4				
49	gyorsulás	uint16	1/2	smart_imu->acc1[0]	2,50	0x7FFF / 5,0	g
50			2/2				
51		uint16	1/2	smart_imu->acc1[1]	2,50	0x7FFF / 5,0	g
52			2/2				
53		uint16	1/2	smart_imu->acc1[2]	2,50	0x7FFF / 5,0	g
54			2/2				
55	mágneses térerősség	uint16	1/2	smart_imu->mag[0]	2,00	0x7FFF / 4	
56			2/2				
57		uint16	1/2	smart_imu->mag[1]	2,00	0x7FFF / 4	
58			2/2				
59		uint16	1/2	smart_imu->mag[2]	2,00	0x7FFF / 4	
60			2/2				
61	repülési mód (manuális/auto)	uint16	1/2	flightmode			
62			2/2				
63	következő útvonalpont	uint16	1/2	nextwaypoint*10+lc			
64			2/2				
65	tartalék hely						
66							
67							
68							
69							
70							
71	EKF státusz	uint16	1/2	state->ms	24		
72			2/2				
73	ellenőrzőösszeg	uint16	1/2	checksum			
74			2/2				

Föld mágneses tere a laborban

A skálázás és offset képzés azért szükséges, hogy az adott szélességen (8, 16, 32 bit) minél több biten legyen ábrázolva egy érték, mivel kis változások esetén a Hamming-távolság¹ kicsi lenne az eredeti számábrázoláson. Ahol szükséges, ott a visszakódolás az alábbi formában történik :

$$\text{eredeti} = (\text{nyers adat/skálazás}) - \text{offset}$$

Az értékek megfelelő kiválasztása a minél nagyobb szétszóráshoz szükséges, érdemes a legnagyobb értékkel elosztani és annak felével eltolni.

2.3. Adatok küldése

A repülőgép által lerepülendő feladat útvonalpontjait hasonlóképpen, mint az adatok fogadását, vezeték nélküli csatornán küldjük fel. A feltöltendő adat küldésének protokollja létfontosságú, mivel ha valamilyen hiba kerül a kommunikációba akkor az akár végzetes is lehet. Gondolok itt olyan hibára, hogy egy fordulópont koordinátája úgy kerül feltöltésre, hogy az kiesik a repülő hatósugarából és ezzel nem számolva, lemerül a tápellátást szolgáló akkumulátor. Az ilyen hibák ellen célszerű a feltöltés protokolljába hibadetektálást építeni, hogy ezek a feldolgozás előtt derüljenek ki.

Felmerül a kérdés, hogy a küldés mikor engedélyezett, a felszállás előtt vagy repülés közben is? Láthattuk néhány megoldásban, hogy lehetőség van az útvonal módosítására menet közben is, ez egy jó opción, így érdemes ezt is megvalósítani. Egyetlen probléma ennek mikéntje, ha csak egy pont koordinátáját módosítjuk, akkor csak ezt vagy az összeset küldjük? Ennek egyik megoldása az lehet, hogy korlátozzuk az először feltöltött pontok számára és az összes pontot újra elküldjük. A módosítás feldolgozását rábízzuk a fedélzeti implementációra, hogyha egy ponton áthaladt, akkor az utólag hiába lett módosítva, a következő fordulópont felé halad. További stratégiák is elképzelhetőek, de a további fejezetekben taglaltak miatt, ezt érdemes választani.

2.3.1. Protokoll

Több megoldás is lehetséges a fordulópontok feltöltésére:

- Rögzített maximális darabszám elküldése egy csomagban
- Változó darabszám esetén egy fordulópont egy csomagban

Az első megoldásban rögzítenénk a fordulópontok maximális számát. Mely azt eredményezné, hogy egy csomagban el lehetne küldeni az egész lerepülendő feladatot. Ha egy pont koordinátájának ábrázolására elég 2^*4 byte, így ha feltételezünk egy 10 pontot tartalmazó (10^*2^*4 byte adat) csomagot, akkor annak mérete fejlécvel (3 bájt), checksum mezővel (2 bájt) 85 bájt. Ehhez hozzájönne még a pontok száma, mely a fogadó oldali feldolgozást segítené, ennek mérete 1 bájt.

Másik lehetőségnél bármennyit (N db) lehetne feltölteni: egy csomag szerkeze: fejléc, küldendő pontok száma, aktuális pont sorszáma, koordinátái, checksum. Ha a küldendő

¹Bináris számok XOR képzésével kapott 1-esek száma

pontok száma és az aktuális pont sorszáma megegyezik és megérkezett minden csomag akkor ACK-val válaszol ha kész a feltöltés. Ez hibakezelés szempontjából kedvezőbb, mivel ha egy pont sorszáma nem egyezik meg az elvárttal, akkor újraküldés kérésével elég csak az adott pont újraküldésével terhelni a csatornát.

Mivel az eddig használt megoldásban a kódba „bele volt égetve” az útvonalterv, mely 5-6 pontot tartalmazott, az első megoldás tűnik kedvezőbbnek. Fogadó oldalon is könnyebb egy ilyen lehetőségre felkészíteni. Ha esetlegesen a jövőben több fordulópontot feltöltésére lesz igény, az is megoldható módosításokkal.

A feltöltés során mindenkettő csatlakoztatott modem segítségével redundánsan küldjük el az előállított csomagot. Ha a csomag sértetlenül megérkezett, ACK jelzéssel válaszolnak, melyet fogadunk és vissza jelezünk a kezelőnek.

Egy 86 bájtos csomag tartalma:

bájt index	leírás	típus	skálázás	offset
0	start	bájt(fixture 'G')		
1	start	bájt(fixture 'P')		
2	start	bájt(fixture 'S')		
3	pontok száma	bájt		
4	pontok[0].lat	uin32	UInt32.MaxValue / 360	180
...				
8	pontok[0].lon	uin32	UInt32.MaxValue / 360	180
...				
12	pontok[1].lat	uin32	UInt32.MaxValue / 360	180
...				
16	pontok[1].lon	uin32	UInt32.MaxValue / 360	180
...				
84	checksum 1/2	uin16		
85	checksum 2/2	uin16		

2.1. táblázat. Küldés protokollja

Felmerülhet a kérdés, hogy a feltöltés ezzel a protokollal elég hibatűrő-e, mivel a fogadás protokollja is hasonló hibadetektálást biztosít, így elégsegesnek tűnik, tovább, ha a küldés megfelelően lezajlott, kapunk vissza jelzést. Ha ez elmaradna, akkor lehetőség van az üzenet újbóli elküldésre.

2.4. Grafikus felület

Az előző fejezetben ismertetett grafikus felületekből levonva a következtetéseket, nyilvánvaló, hogy a GUI kialakításában fontos a repülőgép aktuális pozíójának térképen való mutatása, az repülési állapot könnyen értelmezhető megjelenítése, illetve az esetlegesen előforduló problémák feltűnő jelzése.

2.4.1. Főképernyő

A főképernyőn látható lesz a repülőgép aktuális pozíciója és iránya. A pozicionálást segítendő, egy térkép lesz egy repülőgép ikon háttérben. Ez a rész a képernyő kb. 2/3-át fogja elfoglalni. Az oldalsó sávban a „Glass Cockpit” kerül kialakításra, ez a nézet tartalmazza a gép aktuális sebességét, iránytű segítségével irányát, magasságát, emelkedésének sebességét. Valószínűleg ez a képernyő lesz legnagyobb százalékban használva, így a kritikus hibákról itt kell feltűnő értesítést adni. Melyet a háttérben dolgozó hibadetektáló algoritmus vált ki. Az értesítés egy felugró ablak lenne, mely tartalmazza, mely érték hibájából keletkezett.

2.4.2. Tervezés képernyő

Ezen a képernyőn a felhasználó kijelölheti a lerepülendő útvonalhoz tartozó fordulópontokat, melyet csatlakozás után aszinkron módon feltölthet a repülőre. Mivel a kommunikációs protokoll 10 pontot enged meg, így ennél többet itt ki sem jelölhet, a lerakott pontok helyét a megszokott Google Maps-hoz hasonló módon hosszan kattintva átrakhatónak kell lennie, illetve köztes pontoknak törölhetőeknek kell lenniük. Láthattuk, hogy érdemes a kijelölt útvonal hosszáról tájékoztatni a felhasználót, így ez egy hasznos funkció.

2.4.3. Diagnosztikai képernyő

A 2 porton érkező dekódolt értékek látszódnának 2 oszlopan, mellettük egy hibaérték, mely a különböző hibatípusok hibaszámának összege lenne.

Hibatípusok

- beragadás
- túl nagy változás
- túl nagy különbség a 2 vett értéken

Ezek feldolgozására 2 FIFO sort kell alkalmazni, melyek visszamenőleg tárolják a beérkező értékeket. Ez azért szükséges, mivel így a túlságosan kiugró értékeket detektálni lehet, illetve, ha az egész sorban ugyanazok az értékek vannak, gyanús a beragadás esélye. A harmadik esetben sajnos nem lehetséges a „jó” kiválasztása, mivel nem tudjuk, melyik modemből érkezett adat a megfelelő. Ezt csak háromszorozással és többségi szavazással lehetne megoldani. Így a kettő érték átlagát lehet csak felhasználni.

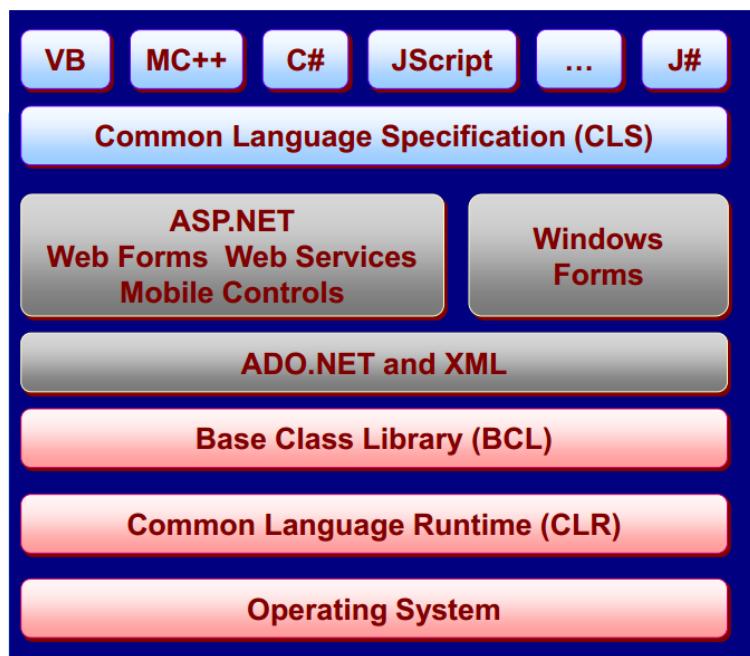
2.4.4. Terminál képernyő

Lehetőség nyílik a fogadott csomagok hexadecimális vagy decimális formában történő megjelenítése, így az operátor alacsony szinten megbizonyosodhat a kapcsolat létrejöttében, mivel láthatja a csomagok beérkezését. Ha a fejléc a csomag elején látszódik, akkor működnie kell a további dekódolásnak, melyet a többi nézet használ fel. Ha nem lát itt adatokat, akkor ellenőrizheti, hogy valóban jó portot illetve adatsebességet választott-e ki.

2.5. Használt technológiák bemutatása

A programot C# nyelven, Microsoft Visual Studio 2010-es verziójával készítem, a használt .NET keretrendszer verziója: 4.5.

2.5.1. .NET



2.1. ábra

A .NET [21] egy menedzselt végrehajtási környezetet biztosító keretrendszer, mely számos szolgáltatást nyújt a benne futtatott programoknak. Két fő részből áll: CLR², mely a programok végrehajtásáért felelős és a .NET BLR³, mely tesztelt, újra felhasználható programkönyvtárakat tartalmaz.

Common Language Runtime

A menedzselt környezetet a CLR [22] biztosítja, a memóriakezelést kiveszi a programozók feladatai közül, mely az egyik legnagyobb odafigyelést igényelte, további feladata a kód végrehajtása, verifikációja, fordítása. Garbage Collector nevű memóriafelszabadítást végző eszköze automatikusan törli a memóriából a már nem hivatkozott elemeket.

Base Class Library

API⁴-k összesége, célja hogy megkönnyítse és gyorsítsa fejlesztés folyamatát. A feladatom megoldásához egyik legfontosabb osztály a *SerialPort*[19] osztály, mely megkönnyíti a soros port kezelését.

²Common Language Runtime (közös nyelvi futatókörnyezet)

³Base Class Library

⁴Application Programming Interface, mely előre megírt komponensek használatához biztosít interfésst

Common Language Specification

A .NET nyelvfüggetlen, így a keretrendszer szolgáltatásai hozzáférhetőek minden nyelv számára, mely nyelvi specifikációnak megfelel. Többek között ezek a nyelvek támogatottak: C#, C++, Visual Basic [23].

Windows Forms

Grafikus megjelenítést biztosít az alkalmazásoknak, különböző elemek (beviteli mező, gomb, kép, választó lista) helyezhetők el rá. GDI⁵ [20] segítségével történik a kirajzolás. A GDI olyan függvények gyűjteménye, amelyek a grafikus elemek (görbék, alakzatok, BMP képek) megjelenítését, szövegek kiíratását teszi lehetővé. A GDI+ ennek továbbfejlesztett változata, mely képes ezen elemek manipulációjához alkalmas mátrixtranszformációkat kezelní és egyéb képformátumok támogatása is megjelent.

C# nyelv

A C/C++ család első valódi objektumorientált tagja, a keretrendszer nagy része C#-ban készült [23]. Más nyelvekkel összehasonlítva tisztább, mint a C++, nagy hasonlóságot mutat a Java nyelvvel.

2.5.2. GMap.NET

A térkép alapú vizualizációhoz egy külső komponenst, a GMap.NET [24] könyvtárcsomagot használom fel. C# nyelven készült, Google Maps-on kívül még számos térkép megjelenítésére képes, API-ján keresztül minden olyan funkció megvalósítható, melyhez a felhasználó hozzászokhatott a Google Maps online felületén. Többek között: útvonalterv összeállítás, pontok kijelölése, .GPX formátumú exportálás, offline üzemmód támogatása. Az én feladatom minden részfeladatát támogatja, ezért is esett erre a csomagra a választásom. A fejlesztője aktív, nagy érdeklődés mutatkozik munkájára, így a talált hibák kijavítása, újítások gyakran feltöltésre kerülnek. Jelenlegi verziója: 1.6, mely Windows Forms, Windows Presentation és Mobile platformokat támogat.

⁵Graphic Device Interface, Microsoft API, mely az operációs rendszer része, feladata grafikus elemek megjelenítése

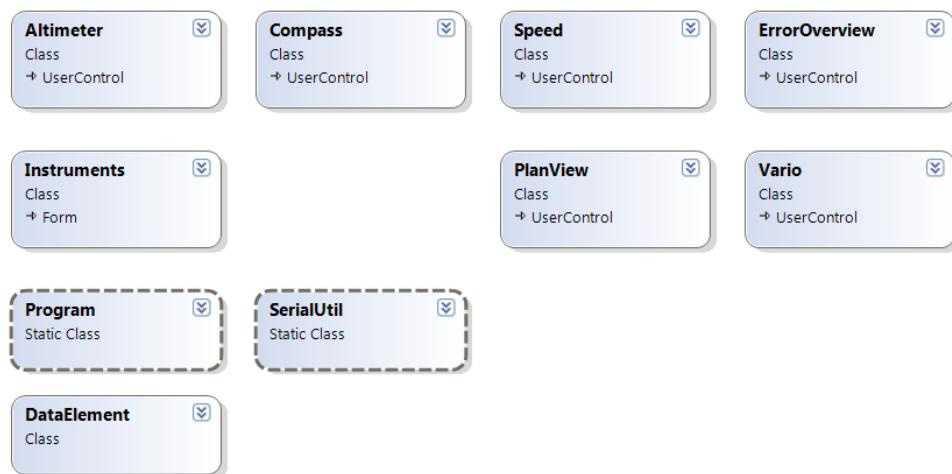
3. fejezet

Megvalósítás

Az előző fejezetekben összegyűjtöttem a megvalósításhoz szükséges információkat, tervezési lépéseket. Ebben a fejezetben a konkrét implementációt fogom bemutatni.

3.1. Program felépítése

A 3.1. ábrán láthatóak a programban használt osztályok. A *Program* statikus osztály a main függvényt, mint belépési pontot tartalmazza. Ez példányosítja az *Instruments* osztályt, mely a különböző *View*-ok megjelenítéséért felelős. Az *Altimeter*, *Compass*, *Speed*, *Vario* a műszereket megvalósító osztályok, a *PlanView* a tervezőképernyő, az *ErrorOverview* osztály a diagnosztikai képernyő implementációja. A soros portból érkező adatokat a *SerialUtil* statikus osztály dolgozza fel, a kapott csomag egy-egy adattagját egy *DataElement* objektumba helyezi, mely a hibadiagnosztikát valósítja meg a benne található elemre.



3.1. ábra

3.1.1. Kapcsolódás megvalósítása

A 3.3 fejezetben biztosítom a program számára a küldő és fogadó oldalt, mely a repülőgépet hivatott helyettesíteni. A kapott adatok soros porton keresztül érkeznek, egy előre meghatározott protokoll szerint (2.2.1 fejezet). A kapcsolódást megkönnyíti az előre elkészített *SerialPort* könyvtár csomag, mely minden szükséges műveletet rendelkezésre bocsájt. Eseményvezérelt műveletei közé tartozik a *DataReceived()* függvény, mely akkor hívódik meg, ha a felépített kapcsolaton keresztül adat érkezett. Jelen esetben a kapcsolat sebességén műlhet, hogy egy ilyen adatcsomagban egy egész számunkra megfelelő csomag érkezett-e. Előfordulhat az az eset, hogy a repülőgép már küldi az adatait és a program ennek az adatfolyamnak a közepébe kapcsolódik bele, így egy előző csomag eleje és a következő csomag vége lemaradhat. Ezért szükséges egy puffer alkalmazni, melynek végére minden egyes beérkező bitet elrak, ha a puffer mérete elérte a fogadandó csomag kétszeresét, biztosak lehetünk abban, hogy ebbe egy csomag már belefér. Ekkor a puffer elejéről egy iteráció elindul, mely az „UUT” fejlécet keresi, ha megtalálta, onnan a megtalált index + csomag hossza indexig iterálva feltölt egy byte tömböt, melyet a *SerialUtil* osztály *Decode()* függvénye fogad.

3.1.2. Redundáns adatok feldolgozása

Mivel párhuzamos csatornákon kapja az adatokat, így az előző fejezetben ismertetett folyamat kétszerezve van, két külön soros portra. Végeredményben a kapott, értelmezhető megfelelő fejléccel kezdődő csomagokat a *SerialUtil.Decode()* függvény alakít át double értékké, mellyel már kényelmesen lehet dolgozni. A dekódolás után minden egyes érték egy hozzá tartozó *DataElement* objektumban tárolódik, az ezeket tartalmazó tömb a 2 port számára közös erőforrás, így a kölcsönös kizárásról gondoskodni kell. Mivel egy tömb írása nem atomi művelet, így a preemptív ütemezéssel ellátott operációs rendszer a portokhoz tartozó szálakat bármikor megszakíthatja, így előfordulhat az az eset, hogy egyik soros portból érkező csomagot a dekódolás után éppen írja az egyik szál, közben a másik porthoz tartozó másik szál is elkezdené írni. Ezt elkerülendő egy Lock objektumon történik a zárolás az írás megkezdésekor, melyet az írás befejezése szabadít fel, míg az objektum zárolt van, a másik szál kénytelen várakozni.

3.1.3. Redundáns adatok hibadetektációja

Minden fogadott adat egy *DataElement* objektumban tárolódik, ebben 2 FIFO lista szerepel, egyik az „A”, másik a „B” portból érkezőknek. Mikor „A”-ból érkezik egy, az *AddA(double item)* függvény teszi bele az „A” FIFO végére, ugyanez a másik portra is érvényes. Mikor egy műszer elkeríti az értéket, melyet mutatni szeretne, akkor a *GetData()* függvényhívással megkapja. A hibakezelés ezen függvényekben van megoldva, minden FIFO sor rendelkezik egy hibaszámlálóval, mely különböző feltételek mellett nő vagy csökken. A *GetData()* függvény aszerint, hogy mely sornak kisebb ez a hibaszámlálója, dönti el, hogy melyikből választaja ki az elemet. A 2.4.3 fejezetben ismertetett hibatípusokat a következő detektációs algoritmusok érzékelik:

Port kiesése: Ha egyik porton érkező adat és érvényes csomag és azt dekódolás után a hozzá tartozó tárolóba teszem, akkor egy számlálót is növelek. Amelyik porton érkezett az adat annak a számlálóját nullázom, így ha csak az egyik portról érkezik adat, akkor csak a másik számlálója növekszik. Ha ez a számláló elérte a FIFO sor méretét, akkor növekszik az adott port hibaszámlálója, ez a port kiesését jelenti.

Beragadás: Egy jelet akkor tekintek beragadt állapotúnak, ha értéke egy bizonyos ideig változatlan marad, jelen esetben a tároló körbefordulási ideje. Ha a sor legutóbbi és a legújabb eleme közti különbség egy ϵ_1 ¹-nál kisebb, akkor növelem a port hibaszámlálóját.

Túl nagy ugrás: Egy új elem érkezésekor a FIFO-ban lévő elemek átlagát kiszámítom és ha az újonnan érkezett érték ϵ_2 -ször nagyobb különbséget mutat, akkor növekszik a hibaszámlálója.

Kettő jel eltérése: A két jel túl nagy eltérésénél nem tudja az algoritmus eldönten, hogy melyik érték a helyesebb, így az eddigi hibaértékek alapján dől el a választás.

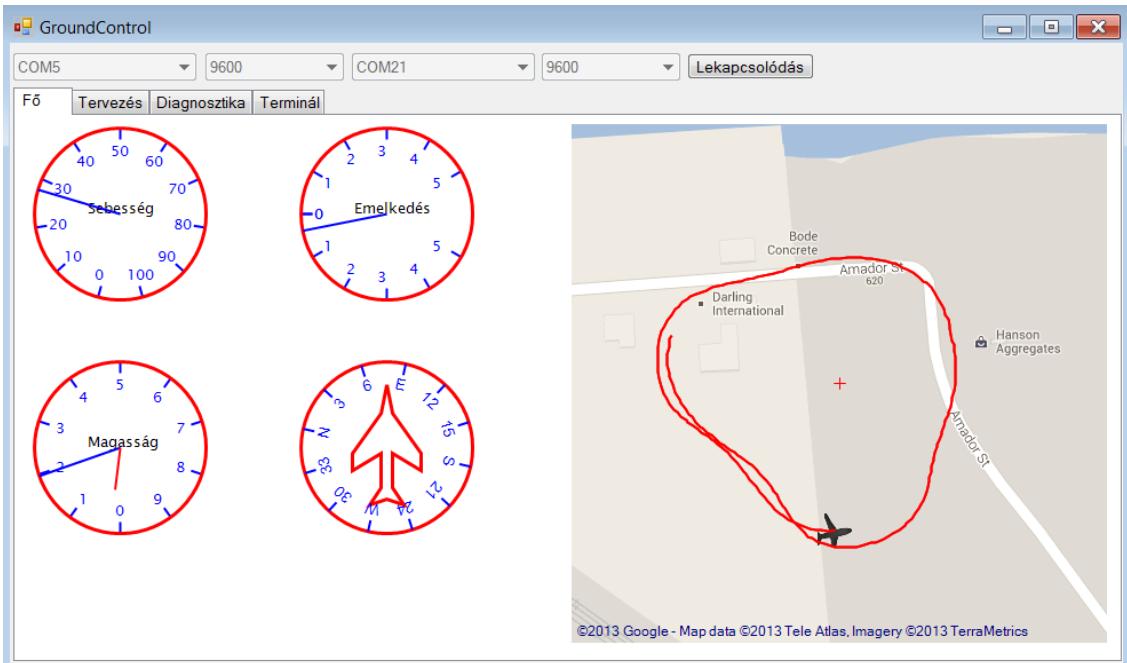
3.2. Megjelenítési felületek

Bal oldalon a létrehozott *UserControl*-ok megjelenítése és elhelyezése a 3.2. ábrán látható. A terület nagyobb részét a térkép foglalja el, melyen a repülőgép aktuális pozíciója látható, illetve a múltban fogadott koordináták tört vonallal összekötve, mely a lerepült útvonalat ábrázolja. A program minimális méretében minden műszer látszódik, a méret módosításával ezek fix pozícióban maradnak, a jobb oldali térkép a jobb alsó sarokhoz van horgonyozva, így ha nagyobb területen szeretné bárki is szemlálni a térképet, akkor látszódik ennek a megoldásnak az előnye.

3.2.1. Főképernyő

A HappyKillmore (1.4.6 fejezet) forráskódjának tanulmányozása során látható, hogy minden műszert külön View-ként valósít meg, így ezt a megoldást célszerű használni. Elkészítettem a magasságmérő, emelkedésjelző, sebességmérő és iránytű *UserControll*-t melyek tervező nézetben „Drag and Drop” technológiával a megfelelő helyre húzható és könnyen beköthető.

¹ez esetben 0.0001



3.2. ábra

Sebességmérő

A *View* egy *UserControll*-ból származik, annak a *OnPaint()* metódusát írom felül, melyben a műszer különböző elemeit pozicionálom a megfelelő helyre. Az *OnPaint()* függvény minden olyan esetben meghívódik, mikor valamely része érvénytelenné, nem láthatóvá válik. Ekkor szükséges az elem újra rajzolása, ez a *View Invalidate()* metódusával explicit is kiváltható. GDI+ segítségével egy piros kört rajzolok

Iránytű

Az iránytű felépítése hasonló, mint a sebességmérő műszernek, jelentős változás az égtájak és a fokok forgatását végző transzformáció. Először a kiírandó karaktert a grafikai koordináta-rendszer középpontjába tolom el, ekkor lehetséges az adott szöggel történő elforgatás, majd a megfelelő helyre eltolás. Ez azt a látszatot kelti, mintha úgy lenne egy tárcsára felírva, hogy az csak akkor olvasható, amikor felső helyzetben van. A kirajzolást végző grafikus megjelenítő pipeline tulajdonsága miatt ezeket a műveleteket fordított sorrendben kell elvégezni a *private void DrawNumbers(Graphics g)* metódusban.

Csatlakozás sáv

A program grafikus felületén a különböző oldalak fülek segítségével válthatóak, ezek felett található a csatlakozáshoz szükséges sáv, mely mindenkorán látszik, hogy éppen csatlakozott-e a program a kiválasztott portokra. Csatlakozás előtt lehetőség van egy lenyíló listából a portokat és a jelsebességeket kiválasztani. Mivel valószínűsíthető, hogy a 2 port azonos sebességgel kommunikál, így az első kiválasztásával a második is átállítódik, persze ha ez az eset mégsem állna fenn, az külön módosítható.

Térképes vizualizáció

A felület jobb oldalán egy Google Maps alapú térkép látható, melyet a GMap.NET API segítségével jeleníték meg. A térképre egy réteget helyeztem el, melyre a fogadott GPS pozíciókat törött vonallal összekötve rajzolom ki a gép által lerepült nyomvonalat, illetve a repülőt szimbolizáló *planeMarker* ikont is a legutóbb kapott pozícióra állítom. Az ezt megvalósító programrészlet:

```
void AddCoordinate(PointLatLng receivedCoordinate)
{
    flightRoute.Points.Add(receivedCoordinate);
    gmap.UpdateRouteLocalPosition(flightRoute);
    planeMarker.Position = receivedCoordinate;
    gmap.Invalidate();
}
```

A *planeMarker* ikont, a fogadott mágneses irány szerint elforgatom, így látszódik a repülő haladási iránya is. A mágneses irányt a kapott Euler szög „psi” értékével állítom be.

A térkép egy lokális cache-ből töltődik be, mely Budapest környékét nagy felbontásban, Magyarországot közepes felbontásban tartalmazza, így a terepen, a program működőképes lesz internet kapcsolat nélkül is. Ha szükséges újabb területeket hozzáadni, az a mellékelt GMapDemo.WindowsForms.exe programmal könnyen megtehető, a generált könyvtárstruktúrával felül kell írni a program által használtat, bővebb információ a F.2 fejezetben található.

3.2.2. Tervezés képernyő

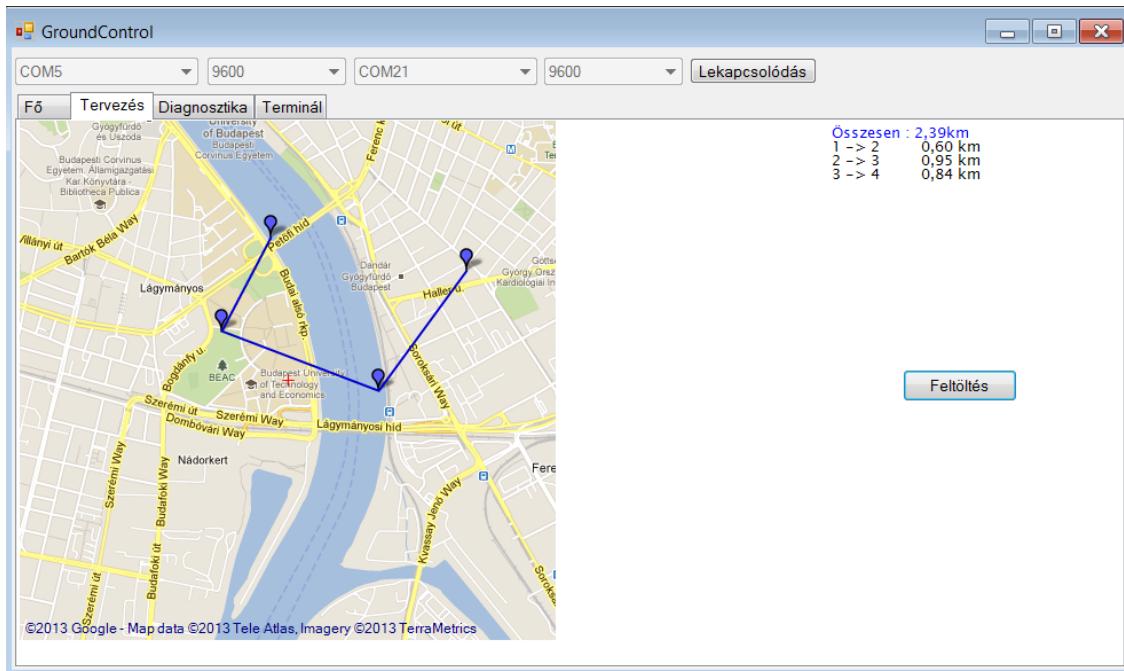
A tervező képernyőn lehetséges az útvonal kijelölése és feltöltése, új útvonalpont a felületen dupla egérkattintással rakható le, mely hosszan nyomva szerkeszthető, jobb kattintással törlhető. A pontokat összekötő szakaszok hosszát jobb oldalt megjelenítem és összegzem. A feltöltés gombra kattintva, az útvonalpontokat tartalmazó listát átadom a SerialUtil.Code() függvénynek, mely a 2.3.1 fejezetben ismertetett protokoll szerint átalakítja bináris formába, ezt az átalakított byte tömböt küldöm el mindenkoros soros portra. Mivel az útvonal maximum 10 db pontot tartalmazhat, ezért új pont lerakásánál ezt szükséges ellenőrizni.

```
private void gmap_plan_MouseDoubleClick(object sender, MouseEventArgs e)
{
    if (plannedRoute.Points.Count < MAX_POINTS && e.Button == MouseButtons.Left)
    {
        PointLatLng currentPos = new PointLatLng(gmap_plan.FromLocalToLatLng(e.X, e.Y).Lat,
                                                gmap_plan.FromLocalToLatLng(e.X, e.Y).Lng);
        GMarkerGoogle marker = new GMarkerGoogle(currentPos, GMarkerGoogleType.blue_small);
        marker.ToolTipText = (numberOfPoints++).ToString();
        planeMarkerOverlay.Markers.Add(marker);
        markerOverlay.Markers.Add(marker);
        plannedRoute.Points.Add(currentPos);
        planView.Invalidate();
    }
}
```

}

}

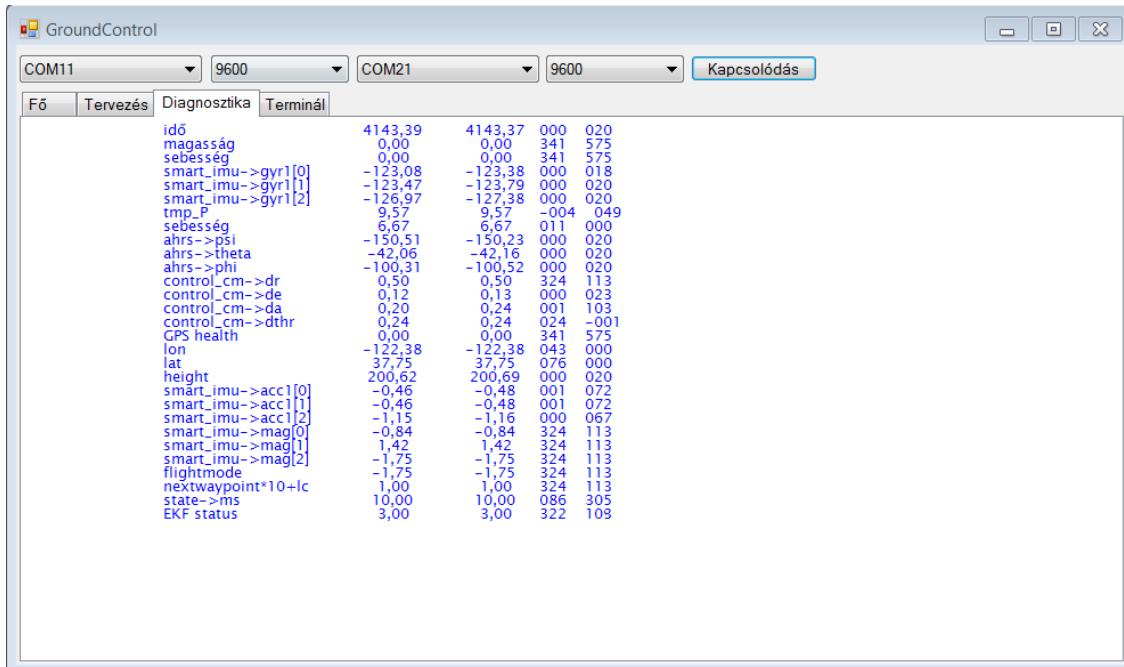
Útvonalpont szerkesztésénél vizsgálni kell, hogy a kurzor aktuális pozíciója egy lerakott pont közelében van-e, ha igen és bal kattintás történik, akkor törölni kell a régi pontot a hozzá kapcsolódó élekkel együtt és új pontot kell létrehozni az aktuális pozícióban.



3.3. ábra

3.2.3. Diagnosztikai képernyő

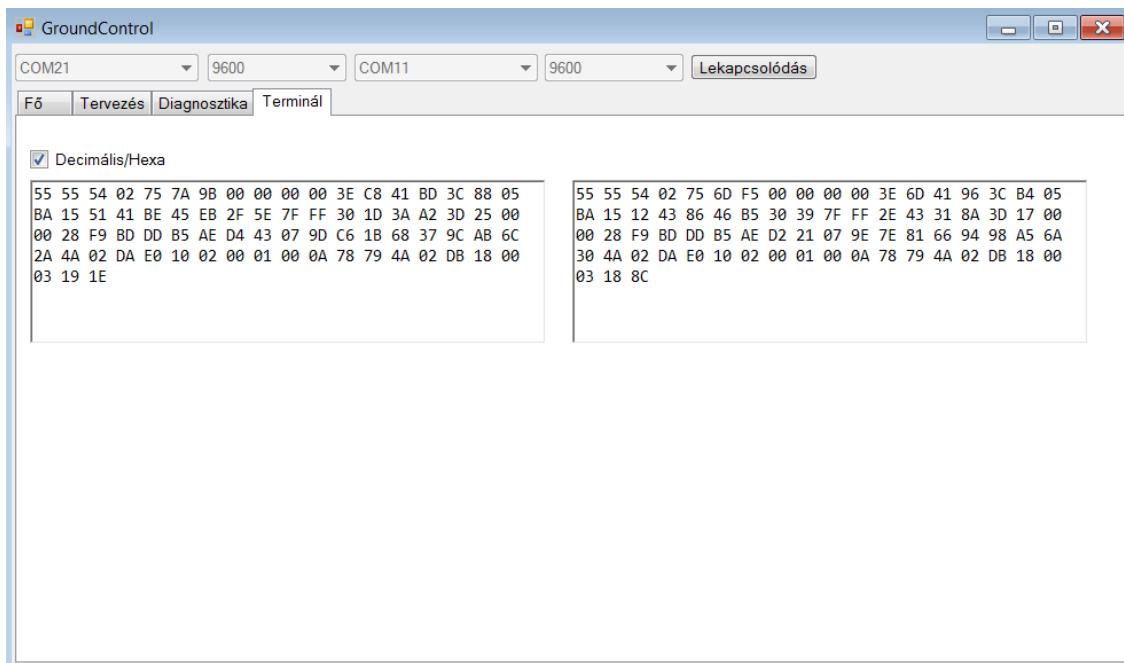
A diagnosztikai képernyőn a portokon érkező adatok és hozzájuk tatozó hibaszámláló láttható. Ha az egyik érték hibaszámlálója elér egy kritikus szintet, akkor pirossá válik és egy felugró ablakon figyelmezteti az operátort, hogy valamilyen hiba történt. Ő eldöntheti, hogy ez csak az egyik műszer meghibásodása vagy esetleg egy fontos összetevő kiesését jelenti. Ennek függvényében mérlegelhet, hogy a repülő folytathatja-e a küldetését vagy érdemes kézi vezérlésre váltani és landolni.



3.4. ábra

3.2.4. Terminál képernyő

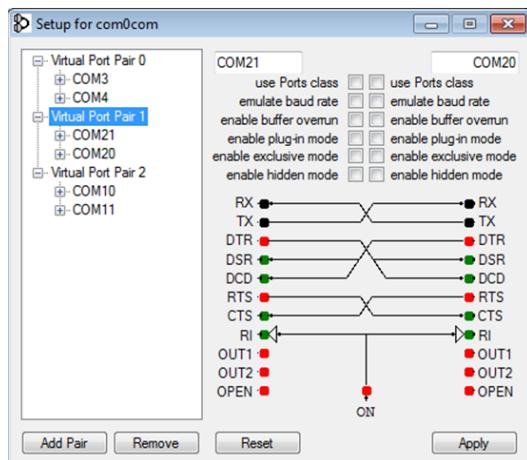
A fogadott csomagokat a terminál képernyő jeleníti meg, ha nem jelennének meg adatok az előző képernyőkön, akkor itt megbizonyosodhat a felhasználó, hogy a program fogadja a csomagokat. Ez a legalacsonyabb szintű kijelzés, ez csak a nyerj bájtokat iratja ki, választhatóan hexadecimális vagy decimális formában. Ha itt sincs változás, érdemes a csatlakoztatott portot vagy a jelsebességet változtatni.



3.5. ábra

3.3. Fejlesztés menete

Az önálló, otthoni fejlesztést megkönnyítendő, a repülőgép HIL² szimulációjából nyert log file-okat használtam fel, így nem volt szükséges ez idő alatt a repülő közelében lennem, elég volt csak a végleges stádiumban kipróbálni a működést. A kapott file-ok felhasználásához először is biztosítani kellett, hogy a fejlesztendő program soros porton keresztül ugyanúgy fogadhassa a csomagokat, mintha azt a repülőgép modemei küldenék. Ehhez készítettem 1-1 programot melyek ezeket a file-okat beolvassák és két kiválasztott soros portra 500 ms-onként egy csomagot küldenek, melyet egy emulátor programmal hoztam létre. Ez az emulátor [25] képes null-modemként viselkedni, melynek lényege, hogy software-es úton biztosít két soros port között kapcsolatot. Így létrehoztam egy COM20-COM21 és egy COM10-COM11 összeköttetést, az egyik program a COM20-ra másik a COM10-re csatlakozik, a fejlesztendő program ezen párok másik portját használja kommunikációra. Mivel szükséges a kétirányú kapcsolat kiépítése, így a log file-okat feldolgozó programba beépítésre került egy beérkező üzeneteket feldolgozó függvény, mely kiírja a konzolos felületére a beérkezett csomagot, így ellenőrizhető, hogy azt küldi-e el amit szeretnék.



3.6. ábra

```
C:\Users\bojtip\Desktop\sorosendB.exe
258*75 bytes
259*75 bytes
260*75 bytes
261*75 bytes
262*75 bytes
263*75 bytes
264*75 bytes
265*75 bytes
266*75 bytes
267*75 bytes
268*75 bytes
269*75 bytes
270*75 bytes
271*75 bytes
272*75 bytes
273*75 bytes
?73*75 bytes
```

3.7. ábra

A kapott log file-ok szerkezetét XVI32 nevű programmal tanulmányoztam és ismertem meg egy csomag felépítését és méretét. Látható, hogy egy csomag mérete valóban a specifikációban megadott 75 byte. Ezt a programot használtam az elküldött útvonalpontok csomagjának vizsgálatához is, hogy valóban úgy került-e elküldésre a csomag, ahogy azt a specifikációban meghatároztam.

²Hardware In the Loop, todo

3.8. ábra

4. fejezet

Összefoglalás

Szakdolgozatom keretében bemutattam a piacon lévő földi irányítóegységek grafikus felületeit, az összehasonlításukból levont követeztetéseket felhasználtam a tervezés fejezetben. Megterveztem, hogy a specifikáció megvalósításához milyen feladatok megoldása szükséges, melyet a megvalósítás fejezetben részletezek. Bemutattam, hogy a részlépések megoldásához milyen eszközöket használtam, hogyan oldottam meg megtervezett funkciókat. Implementáltam egy grafikus felhasználói felületet, melyen a redundánsan küldött rádiójeleket kijelzem. A repülőgép aktuális pozícióját egy Google Maps alapú térképen vizualizáltam. Az párhuzamosan, aszinkron érkező jelek közti eltérést a program detektálja és értesítést ad a kezelő személyzetnek, ha ez elér egy kritikus szintet, továbbá megoldottam az útvonal feltöltésének a lehetőségét, így kényelmesen kijelölhető a repülőgép által bezárandó terület.

Továbbfejlesztési lehetőségeként a grafikus felület finomítását látom, mint látható volt, pár megoldás a beérkező jeleket grafikusan is tudta ábrázolni, mely valószínűleg hasznos lehet ebben a projektben is. Továbbá érdemes lenne a fedélzeten működő hibadetektációs algoritmusuk eredményét is leküldeni, illetve kijelezni, mivel azok nagyobb megbízhatósággal dolgoznak, mint a vett értékek kiértékelésén alapuló.

Irodalomjegyzék

- [1] <http://www.azom.com/article.aspx?ArticleID=10156>, 2013. november 19., 10:00
- [2] <http://www.uvisionuav.com/portfolio/gcs/>, 2013. november 24., 15:00
- [3] http://personal.us.es/imaza/papers/journals/maza_jint10_multimodal/maza_jint10_multimodal_v1.0.1.pdf, 2013. november 24., 15:00
- [4] <http://arduino.cc/>, 2013. december 11., 23:00
- [5] http://hu.wikipedia.org/wiki/CAN_bus, 2013. december 12., 11:00
- [6] <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868>, 2013. október 29., 14:00
- [7] <http://www.makershed.com/v/vspfiles/assets/images/122-32450-xbeetutorial-v1.0.1.pdf>, 2013. december 12., 11:00
- [8] http://www.digi.com/pdf/wp_zigbee.pdf, 2013. november 29., 14:00
- [9] <http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>, 2013. november 29., 15:00
- [10] http://hu.wikipedia.org/wiki/OSI_modell, 2013. december 12., 11:00
- [11] <http://www.sensor-networks.org/?page=0823123150>, 2013. december 02., 22:00
- [12] <https://code.google.com/p/ardupilot-mega/wiki/Mission>, 2013. november 24., 15:00
- [13] <http://paparazzi.enac.fr>, 2013. november 24., 15:00
- [14] <http://www.micropilot.com/products-horizonmp.htm>, 2013. november 24., 15:00
- [15] http://www.szrfk.hu/rtk/kulonszamok/2012_cikkek/77_Makkay_Imre-Papp_Timea.pdf, 2013. november 24., 15:00
- [16] <http://wiki.openpilot.org/display/Doc/OpenPilot+Documentation>, 2013. november 24., 15:00
- [17] <http://qgroundcontrol.org/>, 2013. november 25., 15:00
- [18] <https://code.google.com/p/ardupilot-mega/wiki/HappyKillmore>, 2013. március 19., 16:00

- [19] <http://msdn.microsoft.com/en-us/library/system.io.ports.serialport.aspx>, 2013. március 20, 10:00
- [20] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms533798.aspx>, 2013. december 3, 12:00
- [21] <http://msdn.microsoft.com/en-us/library/hh425099.aspx>, 2013. december 5, 12:00
- [22] <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>, 2013. december 5, 12:00
- [23] <https://www.aut.bme.hu/Course/VIAUA218>, 2013. december 5, 12:00
- [24] <http://greatmaps.codeplex.com/>, 2013. december 8, 11:00
- [25] <http://com0com.sourceforge.net/>, 2013. május 4, 10:00
- [26] <http://www.inf.mit.bme.hu/sites/default/files/materials/category/kategória/oktatás/bsc-tárgyak/rendszermodellezés/13/Hibamodellezés.pdf>, 2013. november 17, 19:00

Függelék

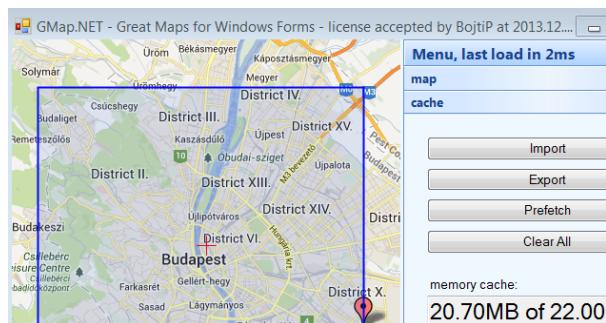
F.1. Program elindítása

Ahhoz, hogy a programot érdemlegesen lehessen futtatni, a 3.3 fejezetben ismertetett programok telepítését írom le.

A program futtatásához Windows XP/Vista/7 és .NET 4.5-s verzió szükséges. Telepítője a mellékleten szerepel *dotNetFx45.exe*¹ néven. A repülőgéppel történő kommunikáció szimulálásához szükséges egy nullmodem telepítése, ezt követően COM10-COMXX és COM20-COMYY párok létrehozása. Windows7 x64 rendszeren a *com0comW7.exe* telepítése, míg XP esetén a *com0comXP.zip* telepítése ajánlott. A COM10 és COM20 portokat fixen a log file beolvasását végző konzolos programok használják, XX és YY helyett szabadon választható. Ha kész az összeköttetés, akkor a *sorossendA.exe* és *sorossendB.exe* futtatásával a log file-ok soros portra íródnak. A *GroundControl.exe* elindításával és a portok kiválasztásával látható a program működése. A kapott log file San Francisco repterén készül HIL szimulációval.

F.2. Térkép letöltése

Új terület letöltésére a *GMap.NET/Demo.WindowsForms.exe* programot lehet használni. Az új területet Alt billentyű hosszan nyomásával és egér mozgatásával lehetséges kijelölni. Ha ez megvan akkor, a „cache” fülön a „Prefetch” gomb hatására letöltődik kiválasztható részletezettséggel, majd az „Export” gombra kattintva megjelenik a mentés útvonala, itt kiválasztva a /map/TileDBv5/en mappában a Data.gmdb file-t, felülírható az offline térképszelvény.



F.2.1. ábra

¹Internet elérés szükséges