



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 14.

# Tartalomjegyzék

<b>Kivonat</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>1. Bevezetés</b>	<b>6</b>
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 . . . . .	14
1.4.1. ArduPilot . . . . .	14
1.4.2. Paparazzi . . . . .	16
1.4.3. MicroPilot Horizon . . . . .	17
1.4.4. OpenPilot . . . . .	19
1.4.5. Egyéb megoldások . . . . .	20
1.4.6. Összehasonlítás . . . . .	21
<b>2. Tervezés</b>	<b>23</b>
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

<b>3. Megvalósítás</b>	<b>30</b>
3.1. Program felépítése . . . . .	30
3.1.1. Kapcsolódás megvalósítása . . . . .	30
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 . . . . .	35
3.2.3. Diagnosztikai képernyő . . . . .	36
3.2.4. Terminál képernyő . . . . .	37
3.3. Fejlesztés menete . . . . .	37
<b>4. Összefoglalás</b>	<b>40</b>
<b>Függelék</b>	<b>43</b>
F.1. Program elindítása . . . . .	43
F.2. Térkép letöltése . . . . .	43

## 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 14.

---

*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. geológiai 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 vizualizáció 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. geological 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 control 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 the route of the plane, on an other window they can see diagnostic data and faults. 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) [1] 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 berepülni (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 feladatuk.

Idővel a technika fejlődésének köszönhetően az UAV-k egyre összetettebb feladatok elvégzésére lettek képesek. A fejlettek rádiótechnika által biztosított nagyobb átviteli sebességek köszönhetően lehetővé vált, hogy a gépet irányító operátor valós időben, moniton 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 feladatok 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 körtségeket. Utasszállító gépek esetében például a robotpilóta elvégez minden olyan korrekciót, amelyhez azelőtt a pilóta folyamatos figyelme, koncentrációja volt szükséges. Ám hiába a fejlett eszközháttér, 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 pozícióját, 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)<sup>1</sup>. 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

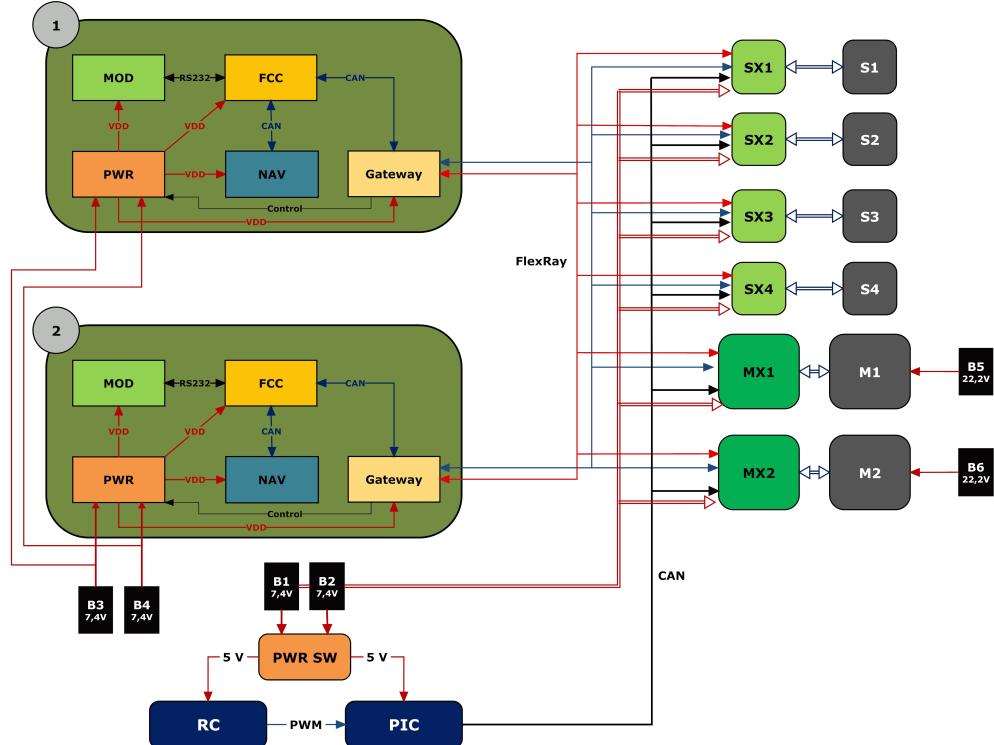
A 1.2. ábrán látható a kialakított architektúra. Elemei:

- **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

---

<sup>1</sup>olyan meghibásodás, mely ha bekövetkezik, az egész rendszer hibás működéséhez vezet

- **Sx, Mx** (Microcontroller) szabályzóelektronika
- **RC** (Remote Control) távirányító, manuális vezérlés
- **PIC** (PIC Microcontroller) PWM<sup>2</sup>-CAN<sup>3</sup> 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)<sup>4</sup>-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-

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

<sup>3</sup>Control Area Network, egy számítógépes hálózati protokoll és adatbusz szabvány melyet mikrokontrollerek és egyéb gazdaszámítógép nélkül működő eszközök kommunikációjára terveztek [2]

<sup>4</sup>orientáció- 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 [3] (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)<sup>5</sup> csomagküldés

**AT:** Alapértelmezetten transzparens módon zajlik a kommunikáció a modemhez csatlakoztatott eszköz és a modem között [4]. 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)<sup>6</sup> jelzéssel bizonyosodhat meg, ennek elmaradásnál újbóli küldés lehetséges. A

<sup>5</sup>előre megírt komponensek használatához biztosít interfész

<sup>6</sup>nyugtázó üzenet, melyet a fogadó küld a feladónak egy csomag sikeres vétele esetén

csomagok többletinformációjával lehetséges broadcast üzenetek<sup>7</sup> 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 [5] 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)<sup>8</sup> modell első (fizikai) és második (adatkapcsolati) rétegét valósítja meg (1.4. ábra), erre építkezik a ZigBee [6] 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)<sup>9</sup> tartományban helyezkedik el.

**A 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.

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

#### **A protokoll jellemzői**

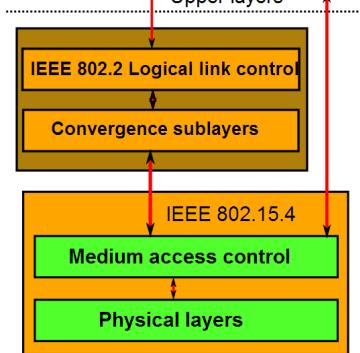
**Alacsony fogyasztás:** A Zigbee protokollra épülő eszközök alacsony fogyasztása abban rejlik, hogy alvó állapotból 30 ms alatt aktívra tudnak váltani, így képeses alacsony fogyasztású állapotban tartózkodni jelentős késleltetés nélkül.

**Nagy hatótávolság:** A 868 MHz-es verzióról a BPSK (Binary Phase-Shift Keying) [8]

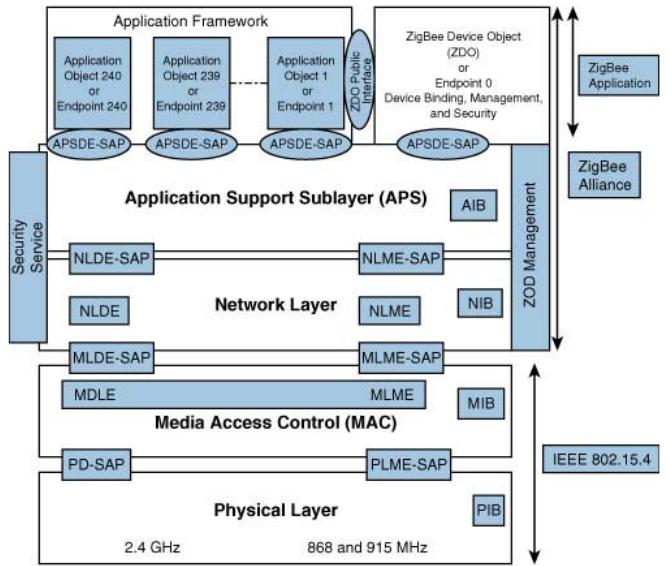
<sup>7</sup>olyan üzenet, melyet egy csomópont az összes vele kapcsolatban álló csomópontnak elküld

<sup>8</sup>rétegekbe szervezett rendszer absztrakt leírása, ami a számítógépek kommunikációjához szükséges hálózati protokollt határozza meg [7]

<sup>9</sup>sávok, 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 GCS (Ground Control Station), ahonnan a földi személyzet irányíthatja és monitorozhatja a repülést. Egy állomás általában több funkciót lát el [9]:

- 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 hatékony 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)<sup>10</sup>-jainak kialakítására. Elsődleges követelmény, hogy az operátor mindenkor a legfontosabb információkat látassa, ehhez szoftverergonomiai alapú megtervezni a grafikus felületet. Kísérleteket folytattak, milyen elrendezésben, hány képernyőn érdemes megjeleníteni az adatokat, úgy, hogy az még ne terhelje túl az operátort [10]. Ebben a kísérletben bebizonyosodott, hogy pl. egy nagy prioritású eseménynél a figyelmeztető ablak megjelenésénél érdemes hangjelzést is adni.

Alábbiakban összehasonlításra kerülnek a piacon lévő megjelenítési felületek, megoldások.

### 1.4.1. ArduPilot

Az Ardupilot projekt [11] létrejöttének oka, hogy otthoni körülmények között, nem ipari alkatrészkből bárki összeállíthatasson egy autonóm járművet, mely az előre betáplált utasításokat végrehajtja. Központi elem az Atmel cég által készített Atmel AVR mikroprocesszorra épülő Arduino platform [12]. Az Arduino egy „system on a board”, aminek lényege, hogy a felhasználó egy előre elkészített eszközt kap, amit egyből a kívánt célra 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 funkcionálitása több nézet segítségével könnyen átlátható. A 1.6. ábrán látható 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.

---

<sup>10</sup>grafikus megjelenítés



1.6. ábra. *MissionPlanner*

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 (1.7. ábra)) 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	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		Lat	Long	Alt	Delete	Up
1	TAKEOFF	0 0 0 0	10,8333060	-14,0625000	100	X			0,00508535...		
2	WAYPOINT	0 0 0 0	4,5654736	63,2812500	100	X			0		
3	RETURN_TO_LAUNCH	0 0 0 0	4,5654736	63,2812500	100	X			nem szám		
4	LAND	0 0 0 0	-25,4829512	3,1640625	100	X			0		

1.7. ábra. *MissionPlanner* tervezőképernyő, pontok típusai

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:

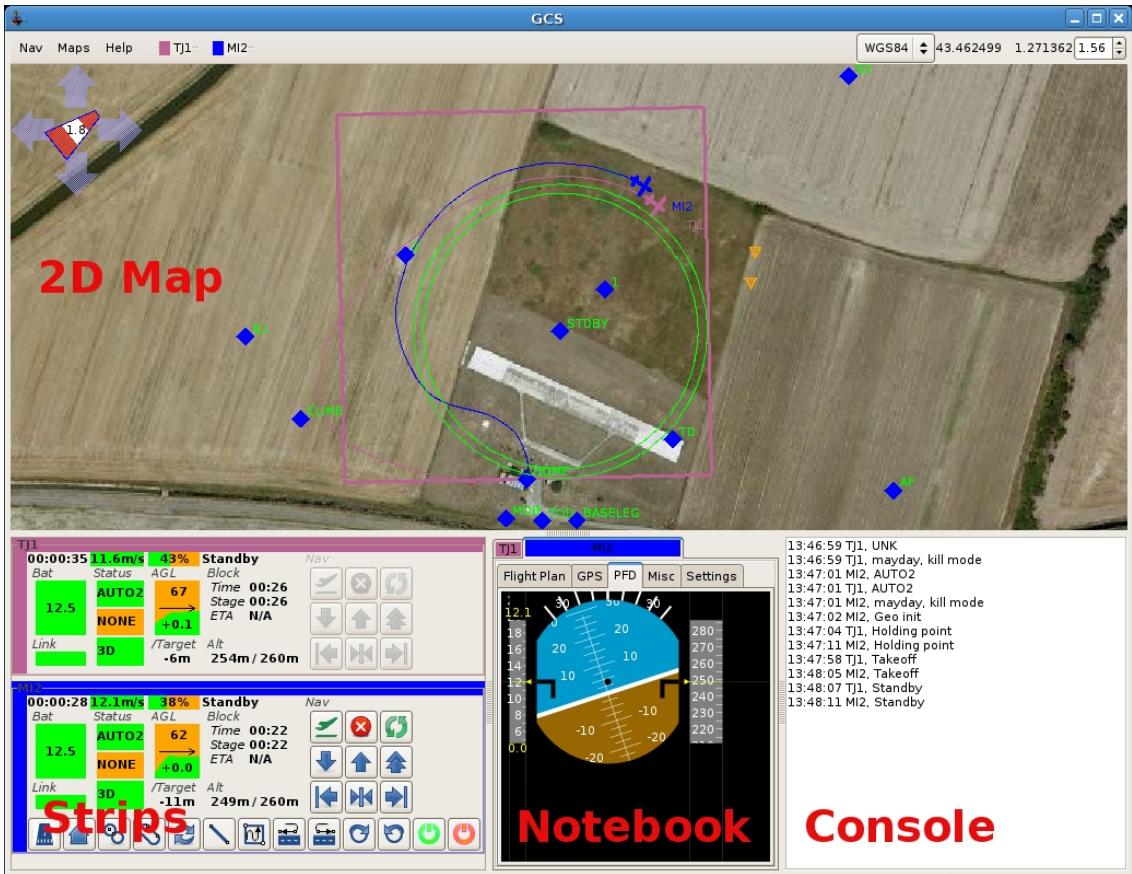
- több platform támogatása (fix- és forgószárny)
- több jármű egyidejű kezelése
- Google Maps, OpenStreetMaps, Microsoft Maps térképes megjelenítése
- küldetéstervezés
- hangjelzés

## 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. *Paparazzi program*

## Összegzés

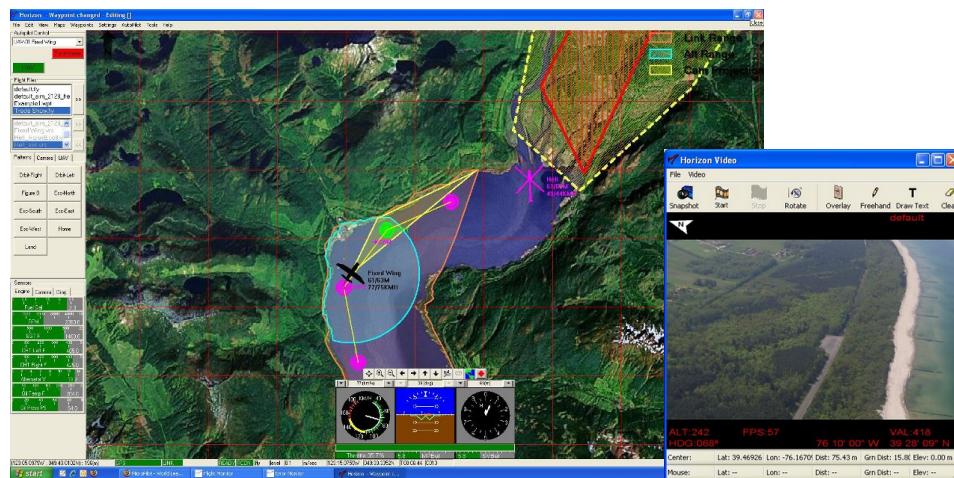
Ugyanaz a képernyő szolgál a navigációra és az útvonal-kijelölésre, ami szerintem nem a legjobb megoldás, mivel nincs elválasztva ez a két funkcionálitás. Egyértelműbb, ha a program a tervezéshez egy olyan perspektívát nyújt, ahol minden szükséges információ rendelkezésre áll az útvonallal kapcsolatban. 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 szervő 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.** *MicroPilot* főképernyője

**1.10. ábra.** *MicroPilot* kamera képe

## 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 (Points Of Interest)<sup>11</sup> gomb, mellyel egy pozíció későbbi kivizsgálásra megjölhető, ha esetleg valami olyat lát az operátor, melyet érdemes felkeresni.

<sup>11</sup>GPS koordinátával megjölhető érdekes hely

#### 1.4.4. OpenPilot

Az OpenPilot projekt [16] célja, hogy nyílt forráskódú, magas minőségű robot pilótát kínáljon autonóm légi járművek vezérléséhez. A projekt 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. Ez a program nem csupán az előbbi programok funkcióival bír (útvonal tervezés, megjelenítés), hanem a fedélzeti számítógépet tartalmazó panel felprogramozására is alkalmas.

#### Főképernyő

A képernyő bal oldalán egy műhorizont látható, alatta a jármű orientációja látszik külső nézetből, míg jobb oldalon a térkép az aktuális pozíciót jeleníti meg.

Ez a nézet szabadon módosítható különféle „gadget”-ek kiválasztásával, pl. a külső nézet egy mért érték grafikus megjelenítésére kicserélhető. Egy .xml fájlba elmenthetőek és visszatölthetőek, így igazodva különböző a konfigurációk igényeihez.



1.11. ábra

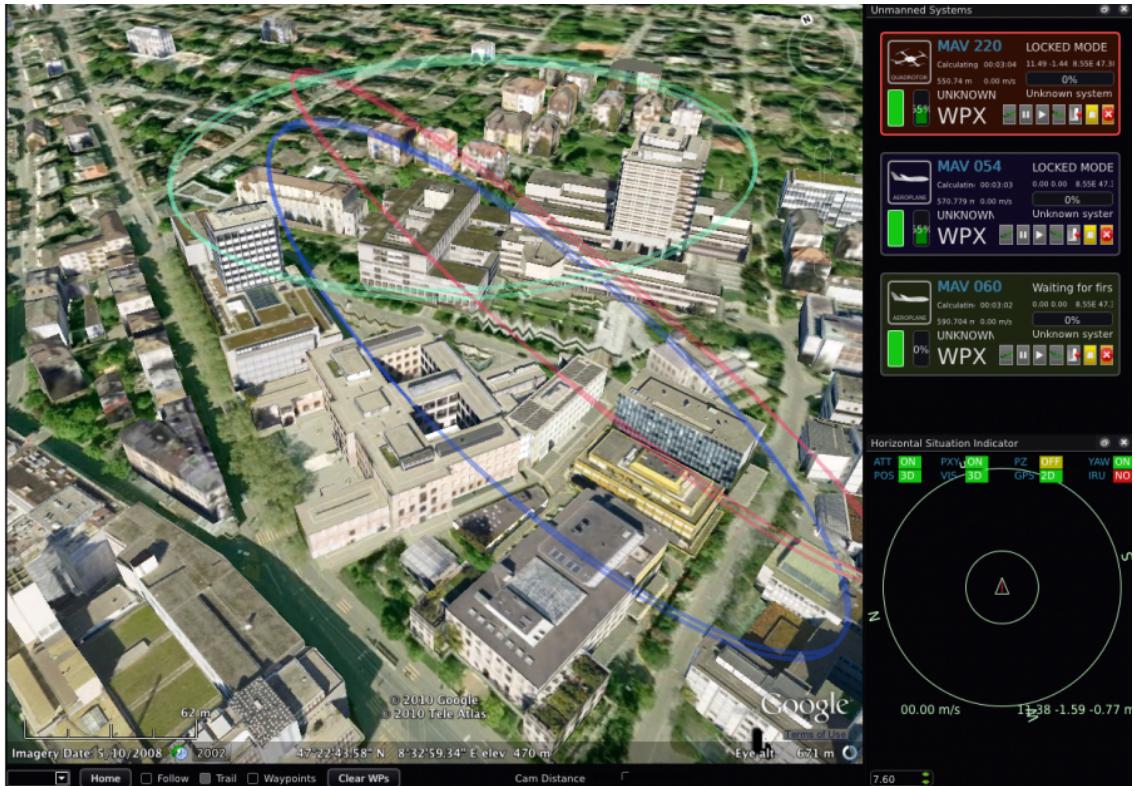
#### Összegzés

A projekt mögött egy igen lelkes és aktív közösség áll, hiszen az OpenPilot minden olyan fontos igényt kielégít, mely napjainkban felmerülhet egy UAV adatainak megjelenítésével és konfigurálásával kapcsolatban.

#### 1.4.5. Egyéb megoldások

##### QGroundControl

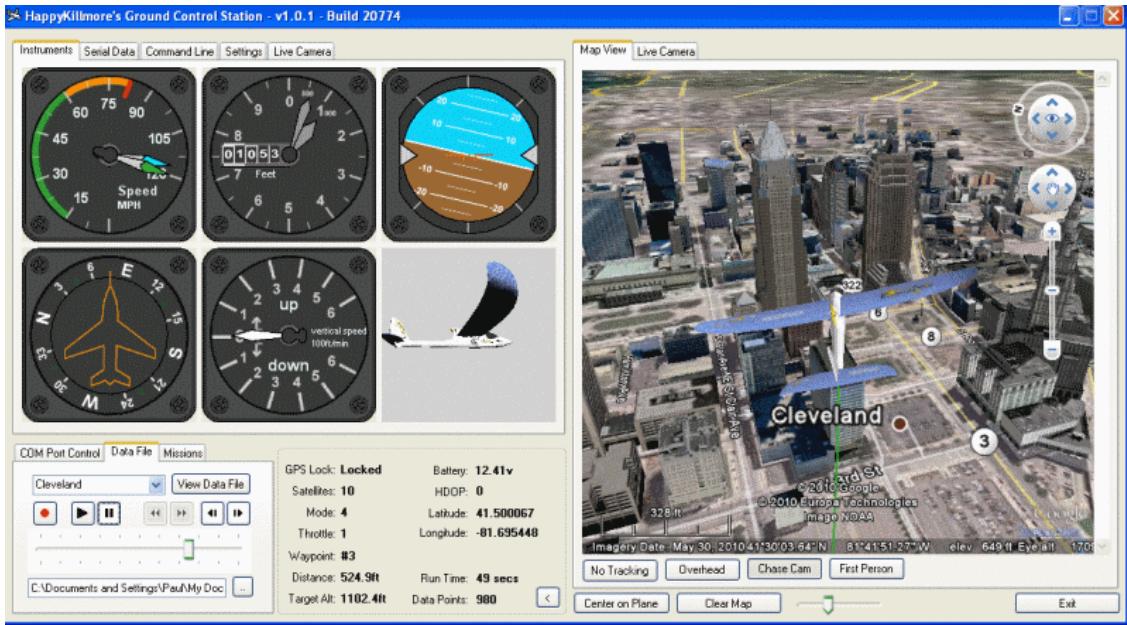
Ez a program [17] nyílt forráskódú és a MAVLink protokollt használja a csatlakoztatott eszközzel történő kommunikációra. Érdekesség, hogy a program a repülőgép aktuális pozícióját a 1.12. látható módon 3D-ben is meg tudja jeleníteni és videókép fogadására is alkalmas. Az útvonalpontok szerkesztése felszállás után is lehetséges. A protokollnak köszönhetően a program képes 255 jármű egyidejű kezelésére.



1.12. ábra. *QGroundControl* program

##### HappyKillmore

Az ArduPilot-hoz készült nyílt forráskódú GCS [18] a Mission Planner-hez képest inkább az egyszerűség híve. Kevés műszer található rajta, de ezekről minden információ megtudható. Itt is a térkép nézet dominál, a képernyő oldalsó részén mozgathatjuk számunkra megfelelő elrendezésbe a műszereket. Útvonalpontok feltöltésére szintén van lehetőség. A program támogatja többek között az ArduPilot és a MAVLINK protokollokat.



1.13. ábra. HappyKillMore program

#### 1.4.6. 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

<b>Program</b>	<b>Ardupilot</b>	<b>Paparazzi</b>	<b>MicroPilot</b>	<b>QGroundControl</b>	<b>HappyKillMore</b>
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 lépéseit, illetve az előző alkotásokból levont következtetéseket és megoldásokat felhasználom.

A következő fejezetben leírom, hogy miket szükség megoldani a program megvalósítása során.

### 2.1. Kommunikáció megvalósítása

Az elkészítendő program (GroundControl) és a repülőgép között kommunikáció csatornánként 2 db modem segítségével történik, a modemek egymás közt vezeték nélkül csatlakoznak, felhasználói oldalon soros portot biztosítanak. Így a 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 kezeln a két párhuzamos csatornán kapott adatokat. Mivel a csomagok aszinkron módon érkeznek, ezért 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 következő:

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. Amennyiben nem egyezik a fogadott checksum-mal, akkor az egész csomag eldobásra kerül.



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-távolság<sup>1</sup> 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, ezért érdemes a legnagyobb értékkel elosztani és annak felével eltolni.

### 2.3. Adatok küldése

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

Felmerül a kérdés, hogy az adatküldés mikor engedélyezett, a felszállás előtt vagy repülés közben is? A bemutatott földi állomások közül néhány lehetőséget biztosított az útvonal repülés közbeni módosítására. Ennek a megoldásánál dönten kell, hogy ha csak egy pont koordinátája módosul, akkor csak az vagy az összes újraküldésre kerüljön-e? Egyik megoldási lehetőség csak a módosított pontot elküldeni és annak feldolgozását a fedélzeti implementációra bízni. Ha a repülő egy ponton áthaladt és az utólag került módosításra, figyelmen kívül hagyja és folytatja útvonalát, viszont ha egy előtte lévő módosult, akkor az újat követi. További stratégiák is elképzelhetőek, a következő fejezetekből kiderül, melyiket 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 rögzített a fordulópontok maximális száma. Ez azt eredményezné, hogy egy csomagban el lehet küldeni az egész lerepülendő feladatot. Ha egy pont koordinátájának ábrázolására  $2^*4$  bájt szükséges és felépíték egy 10 pontot tartalmazó ( $10^*2^*4$  bájt adat) csomagot, akkor annak mérete fejlécettel (3 bájt), checksum mezővel (2 bájt) 85 bájt. Ehhez hozzájön még a pontok száma (1 bájt), ami a fogadó oldali feldolgozáshoz szükséges.

Másik lehetőségnél bármennyi pont feltöltése lehetséges, ez esetben egy csomag a következőket tartalmazza: fejléc, küldendő pontok száma, aktuális pont sorszáma, koordinátái,

---

<sup>1</sup>Bináris számok XOR képzésével kapott 1-esek száma

checksum. Amennyiben a küldendő pontok száma és az aktuális pont sorszáma megegyezik és minden csomag megérkezett, akkor ACK-val válaszol a fogadó fél, jelezve, hogy kész a feltöltés. Ez hibakezelés szempontjából kedvezőbb, hiszen 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 a fedélzeteti programot.

A feltöltés során mindenkettő csatlakoztatott modem segítségével redundánsan küldöm el az előállított csomagot. Ha a csomag sérültlenül megérkezett, ACK jelzést küld a repülőgép, amit fogadva a GroundControl visszajelzést ad a kezelőnek.

Egy 86 bájtos csomag felépítése a következő:

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 ez a megoldás elégsgesnek tűnik. Továbbá, ha a küldés sikereségének visszajelzése elmaradna, akkor lehetőség van az üzenet újbóli elküldésére.

## 2.4. Grafikus felület

Az előző részben ismertetett grafikus felületekből látszik, hogy a GUI kialakításában fontos a repülőgép aktuális pozíójának térképen való mutatása, a repülési állapot könnyen értelmezhető megjelenítése és az esetlegesen előforduló problémák feltűnő jelzése.

#### **2.4.1. Főképernyő**

A GroundControl főképernyőn látható a repülőgép aktuális pozíciója és iránya. A térképen a repülőgép pozícióját egy repülőgép ikon szemlélteti. Ez a rész a képernyő kb. 2/3-át foglalja el. 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 képernyőt fogják a leggyakrabban használni, így a kritikus hibákról itt kell feltűnő értesítést adni, melyet a háttérben dolgozó hibadetektáló algoritmus vált ki. Az értesítés egy felugró ablak, ami mutatja, mely érték hibájából keletkezett.

#### **2.4.2. Tervezés képernyő**

A tervezés képernyőn a felhasználó kijelölheti a lerepülendő útvonalhoz tartozó fordulópontokat, melyeket csatlakozás után aszinkron módon feltölthet a repülőre. Mivel a kommunikációs protokoll 10 pontot enged meg, így ennél többet itt ki sem jelölhet. Az operátor a lerakott pontok helyét a megszokott Google Maps-hez hasonló módon hosszan kattintva átrakhatja, illetve köztes pontokat törölheti. A már létező megoldások összehasonlításából kiderült, hogy érdemes a kijelölt útvonal hosszáról tájékoztatni a felhasználót.

#### **2.4.3. Diagnosztikai képernyő**

A diagnosztikai képernyőn a két porton érkező dekódolt értékek látszódnak két oszlopból, mellettük egy hibaérték, mely a különböző hibatípusok hibaszámának összege.

#### **Hibatípusok**

- beragadás
- túl nagy változás
- túl nagy különbség a két vett érték között

Ezen hibák feldolgozására két FIFO sort érdemes alkalmazni, melyek visszamenőleg tárolják a beérkező értékeket. Ez azért szükséges, mert 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, de mivel jelen esetben ez a lehetőség nem áll fenn, így a kisebb hibaértékű adtot használja fela program.

#### **2.4.4. Terminál képernyő**

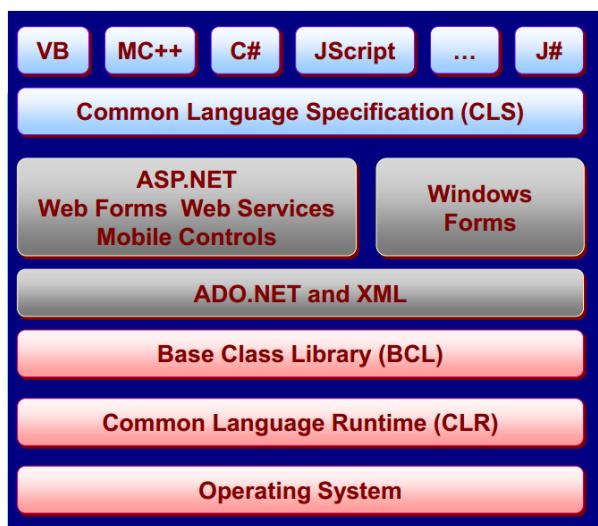
Ebben a nézetben lehetőség nyílik a fogadott csomagok hexadecimális vagy decimális formában történő megjelenítésére, így az operátor alacsony szinten (a fogadott csomagok fel-dolgozatlan formájából) megbizonyosodhat a kapcsolat létrejöttéről, mivel láthatja, hogy csomagok beérkeznek-e. Ha a fejléc a 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 készíttem, a használt .NET keretrendszer verziója: 4.5.

### 2.5.1. .NET



2.1. ábra

A .NET [20] egy menedzselt végrehajtási környezetet biztosító keretrendszer, mely számos szolgáltatást nyújt a benne futtatott programoknak. Két fő részből áll: CLR (Common Language Runtime)<sup>2</sup>, mely a programok végrehajtásáért felelős és a .NET BCL (Base Class Library), mely tesztelt, újra felhasználható programkönyvtárakat tartalmaz.

#### Common Language Runtime

A menedzselt környezetet a CLR [21] biztosítja, a memóriakezelést kiveszi a programozók feladatai közül, mely eddig az egyik legnagyobb odafigyelést igényelte. További feladatai a kód végrehajtása, verifikációja és fordítása. Garbage Collector nevű memória-felszabadítást végző eszköze automatikusan törli a memoriából a már nem hivatkozott elemeket.

#### Base Class Library

API-k (Application Programming Interface)<sup>3</sup> összesége, célja hogy megkönnyítse és gyorsítsa a fejlesztés folyamatát. A feladatom megoldásához az egyik legfontosabb osztály a *SerialPort*[19] osztály, mely megkönnyíti a soros port kezelését.

<sup>2</sup>közös nyelvi futtatókörnyezet

<sup>3</sup>mely előre megírt komponensek használatához biztosít interfész

## **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 [22].

## **Windows Forms**

Grafikus megjelenítést biztosít az alkalmazásoknak, különböző elemek (beviteli mező, gomb, kép, választó lista) helyezhetők el rá. GDI (Graphic Device Interface)<sup>4</sup> [23] 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, új képformátumokat is támogat.

## **C# nyelv**

A C/C++ család első valódi objektumorientált tagja, a keretrendszer nagy része C#-ban készült [22]. Más nyelvekkel összehasonlítva tisztább, mint a C++ és nagy hasonlóságot mutat a Java nyelvvel.

### **2.5.2. GMap.NET**

A térkép alapú vizualizációhoz egy külső komponenst, a GMap.NET [24] könyvtárcomagot használom fel. C# nyelven készült, Google Maps-en kívül még számos térkép megjelenítésére képes. API-ján keresztül minden olyan funkció megvalósítható, melyhez a felhasználó hozzászokhatott a Google Maps online felületén, például útvonalterv összeállítás, pontok kijelölése. Továbbá hasznos funkciói a .GPX formátumú exportálás és az offline üzemmód támogatása.

A fejlesztője aktív, nagy érdeklődés mutatkozik munkájára, így a talált hibák kijavítása és az újítások gyakran feltöltésre kerülnek. Jelenlegi verziója: 1.6, mely Windows Forms, Windows Presentation és Mobile platformokat támogat.

A programom térképes megjelenítéséhez minden funkciót támogat, ezért is esett erre a csomagra a választásom.

---

<sup>4</sup> Microsoft API, mely az operációs rendszer része, feladata grafikus elemek megjelenítése

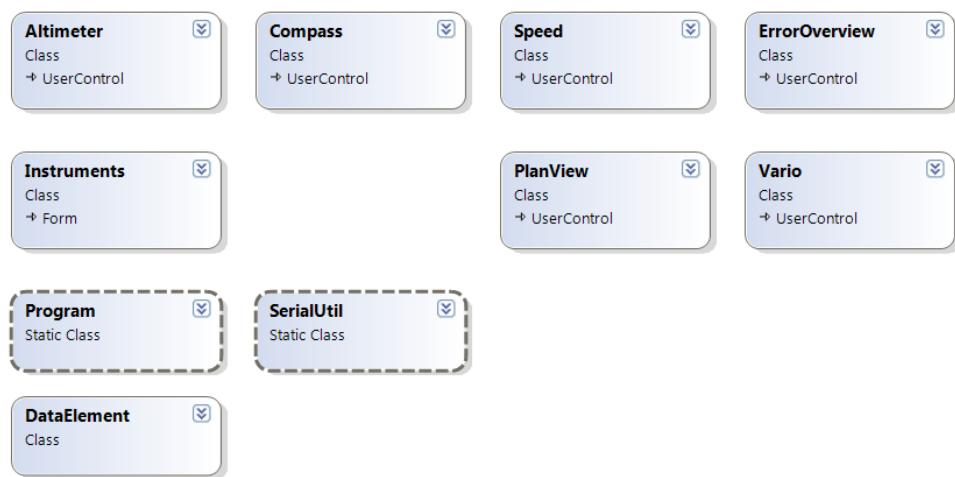
### 3. fejezet

## 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* osztály a tervezőképernyő, míg az *ErrorOverview* osztály a diagnosztikai képernyő implementációja. A soros portból érkező adatokat a *SerialUtil* statikus osztály dolgozza fel, a kapott csomag egy-egy adattaggát egy *DataElement* objektumba helyezi, mely egyben a hibadiagnosztikát is megoldja.



3.1. ábra. Felhasznált osztályok

#### 3.1.1. Kapcsolódás megvalósítása

A kapcsolódást megkönnyíti az előre elkészített *SerialPort* könyvtár csomag, mely minden szükséges műveletet rendelkezésre bocsát. Eseményvezérelt műveletei közé tartozik a *Da-*

*taReceived()* függvény, mely akkor hívódik meg, ha a felépített kapcsolaton keresztül adat érkezett. Jelen esetben a kapcsolat sebességén múlhat, 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 bájt elraktározódik. 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 a program párhuzamos csatornákon kapja az adatokat, így az előző fejezetben ismertetett folyamat kétszerzve 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 két 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. Ebben az esetben előfordulhat, hogy az egyik soros portból érkező csomagot a dekódolás után az egyik szál éppen írja a csomaghoz tartozó *DataElement-be*, miközben a másik porthoz tartozó szál is elkezdené írni ugyanezt. Ezt elkerülendő az írás megkezdésekor egy *Lock* objektumon történik a zárolás, melyet az írás befejezése szabadít fel. Amíg ez az objektum zárolt, a másik szál kénytelen várakozni.

### 3.1.3. Redundáns adatok hibadetektációja

Minden fogadott adat egy hozzá tartozó *DataElement* objektumban tárolódik, ezekben két FIFO lista szerepel, egyik az „A”, másik a „B” portból érkezőknek. Ha az „A” portból érkezik egy csomag, akkor dekódolás után az elemein végigiterálva a *dataElements[i].AddA(double item)* függvény meghívásával kerülnek az „A” FIFO végére, ugyanez a másik portra is érvényes.

Mikor egy műszer elkéri az értéket, melyet mutatni szeretne, akkor a *GetData()* függvényhívással megkapja. A hibakezelés ebben függvényekben van megoldva, ugyanis minden kettő 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 melyiket választja. A 2.4.3 fejezetben ismertetett hibatípusokat a következő detektációs algoritmusok érzékelik:

**Port kiesése:** Mikor az egyik porton egy érvényes csomag érkezik, akkor dekódolás után az elemeit a hozzájuk tartozó tárolóba teszem és abban egy számlálót is növelek. Amelyik porton érkezett az adat annak a számlálóját nullázom, így ha csak az egyik portról érkezik

adat, akkor a másik számlálója növekszik. Ha ez a számláló elérte a FIFO sor méretét, akkor növekszik az adott adattag hibaszámlálója, mely a hozzá tartozó port kiesését jelenti.

**Beragadás:** Egy jelet akkor tekintek beragadt állapotúnak, ha értéke egy bizonyos ideig változatlan marad, jelen esetben a tároló feltöltődésének ideje ideje. Ha a sor legutóbbi és a legújabb eleme közti különbség egy  $\epsilon_1$ -nél (ez esetben 0.0001) kisebb, akkor növelem a jel hibaszámlálóját.

**Túl nagy ugrás:** Egy új elem érkezésekor a FIFO-ban lévő elemek átlagát kiszámítom és ha az újonnan érkezett érték  $\epsilon_2$ -ször (ez esetben 0.01) nagyobb különbséget mutat, akkor növekszik az érték és port 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.

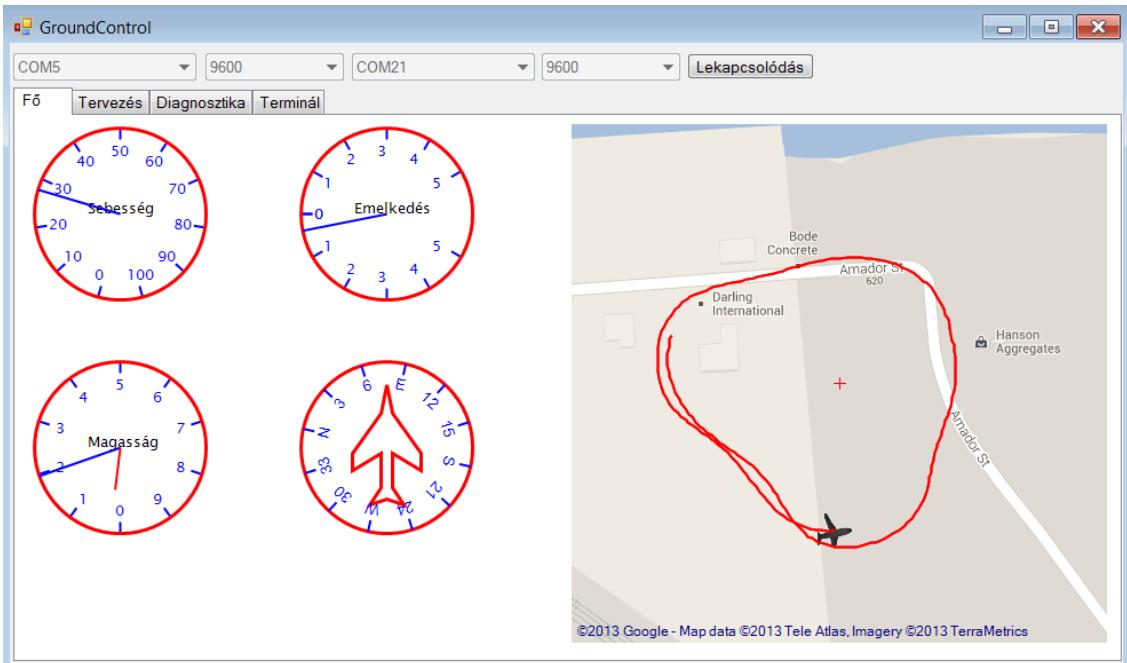
### 3.2. Megjelenítményi felületek

Felhasználói szempontból a grafikus megjelenítés a legérdekesebb, ez ha nincs megfelelően elkészítve a fentebb tárgyalt megoldások akár értelmüket is veszthetik. Ezért a felület kényelmes kezelhetőségére és az ablakok elhelyezésére ügyelni kell. A továbbiakban az elkészített képernyőket mutatom be.

#### 3.2.1. Főképernyő

A HappyKillmore (1.4.5 fejezet) forráskódjának tanulmányozása során láttam, hogy az alkotók minden műszert külön View-ként valósítottak meg. Mivel ez átláthatóbbá teszi a műszerek funkcióját ezt a megoldást célszerű használnom. Elkészítettem a magasságmérő, emelkedésjelző, sebességmérő és iránytű *UserControl*-jait melyek tervező nézetben „Drag and Drop” technológiával a megfelelő helyre húzhatóak és könnyen bekötethetők.

A 3.2. ábrán látható képernyő bal oldalán látható a létrehozott *UserControl*-ok megjelenítése és elhelyezése. A terület nagy részét a térkép foglalja el, melyen a repülőgép aktuális pozíciója látható. A fogadott koordináták tört vonallal vannak összekötve, mely a lerepült útvonalat ábrázolja. A program minimális méretében minden műszer megfelelően látszik, a méret módosításával ezek fix pozícióban maradnak. A térkép a program jobb alsó sarkához van horgonyozva, így átméretezéskor a térkép területe is változik.



**3.2. ábra.** *GroundControl főképernyője*

### Sebességmérő

A sebességmérő 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 újrarajzolása, ez a *View Invalidate()* metódusával explicit is kiváltható. GDI+ segítségével egy piros kört rajzolok, egy iterációval 36°-onként egy sugárirányú szakaszt rajzolok és tízesével növelve kiíratok egy számot, mely a sebességértéket jelöli. Az osztály tartalmaz egy **private float maxSpeed;** változót, ez jelöli a kijelezendő legnagyobb sebességet (most 100 km/h). A **private float currentSpeed;** változó értéke alapján egy mutató mutatja a beállított aktuális sebességet. Ez a kör középpontjából indul és a **currentSpeed / (maxSpeed - minSpeed)** összefüggésből származó szöget zár be a 0 km/h-t jelölő ponttal.

### 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 egy tárcsára felírva, ami csak felső helyzetben olvasható. A kirajzolást végző grafikus megjelenítő pipeline tulajdonsága miatt ezeket a műveleteket fordított sorrendben kell elvégezni a *private void DrawNumbers(Graphics g)* metódusban:

```
//measure the size of the degree, which is written
SizeF textSize = g.MeasureString(degree, this.Font);
```

```

//calculate the radian of i.th iteration's degree
float angle2 = -CalculateRadian((float)i * 30+(float)heading +180);
g.DrawLine(new Pen(Color.Blue, 2), center.X + (float)((size / 2) - 10)
    * (float)Math.Sin(angle2), center.Y + (float)((size / 2) - 10) *
    (float)Math.Cos(angle2), center.X + (float)(size / 2) *
    (float)Math.Sin(angle2), center.Y + (float)(size / 2) *
    (float)Math.Cos(angle2));
//translate to the proper position
g.TranslateTransform(center.X + (float)((size / 2) - 20) *
    (float)Math.Sin(angle2) - 0, center.Y + (float)((size / 2) - 20) *
    (float)Math.Cos(angle2) - 0);
//rotate with the needed degree
g.RotateTransform(i * 30 + (float)heading + 180 + 180);
//translate to the center of the string
g.TranslateTransform(-stringSize.Width / 2, -stringSize.Height / 2);
g.DrawString(degree, new Font("Arial", 8), new SolidBrush(Color.Blue),
    0F, 0F);
g.ResetTransform();

```

## Csatlakozás sáv

A program grafikus felületén a különböző oldalak fülek segítségével válthatóak. Ezek felett található a csatlakozáshoz szükséges sáv, mely mindenkor látszik, hogy éppen csatlakozott-e a program a kiválasztott portokra. Csatlakozás előtt lehetőség van egy lenyíló listából a portokat és a jelszavakat kiválasztani. Mivel valószínűsíthető, hogy a két 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 jelenít meg. A térképre egy réteget helyeztem el, melyre a fogadott GPS pozíciókat törött vonallal összekötve rajzolom ki a gép által lerepült nyomvonalaat. Erré a rétegre kerül a repülőt szimbolizáló *planeMarker* ikont is, aminek pozícióját mindenkor legutóbb kapott koordináta határozza meg. 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”<sup>1</sup> értékével állítom be.

A térkép egy lokális cache-ből töltődik be, mely Budapest környékét nagy felbontásban, Magyarországot közepes felbontásban tartalmazza. Ennek segítségével a térkép működőképes internet kapcsolat nélkül is. Ha szükséges újabb területek hozzáadása, az a mellékelt GMapDemo.WindowsForms.exe programmal segítségével lehetséges. A generált fájllal felül kell írni a program által használtat, további tudnivaló a F.2 fejezetben található.

### 3.2.2. Tervezés képernyő

A tervező képernyőn (3.3. ábra) lehetséges az útvonal kijelölése és feltöltése. Új útvonalpont a felületen dupla egérkattintással rakható le, mely hosszan nyomva szerkeszthető, jobb kattintással törölhető. A pontokat összekötő szakaszok hossza jobb oldalt megjelenik és a tervezhetőség kedvéért összegzsre is kerül.

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 bájt tömböt küldöm el mindenkor soros portra, végső soron a repülőgépre. 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();
    }
}
```

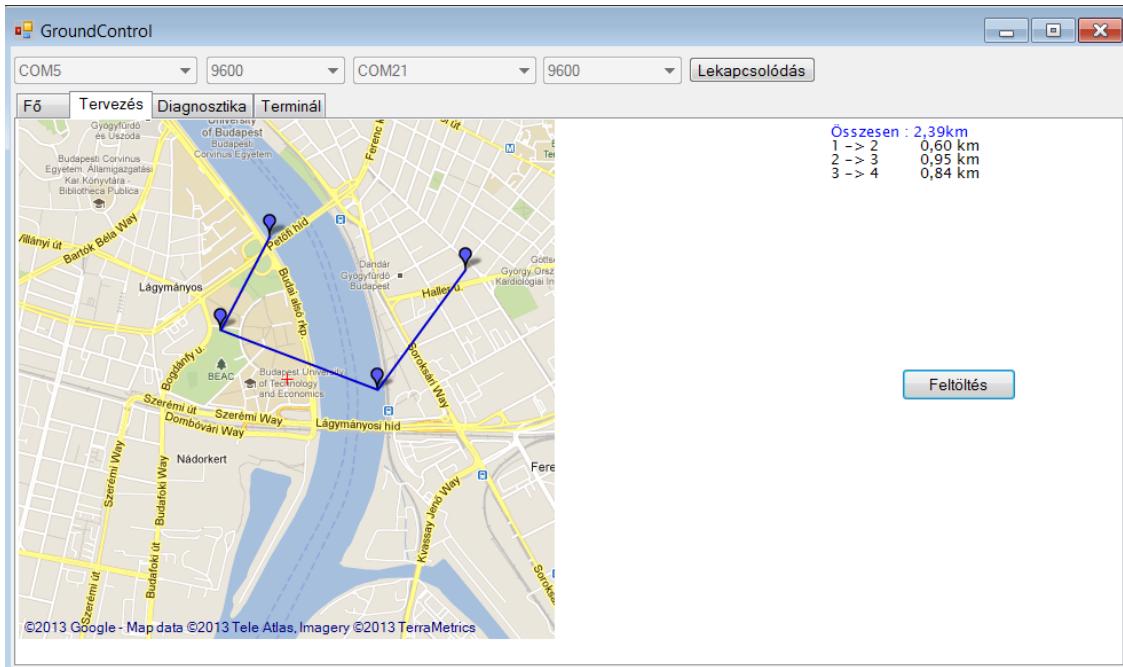
Ha a feltöltött lista sértetlenül megérkezett, a repülőgép az aktuális kommunikációs csatornáján egy „ACK” üzenettel jelez. Ezt fogadva az operátort egy tájékoztató ablak formájában (3.4. ábra) tájékoztatja a program.

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. Ezt új ponthoz az előtte és utána lévő pontokkal össze kell kötni. Így egy pont újrapozicionálása

---

<sup>1</sup>a repülőgép függőleges tengelye körüli elfordulás mértéke a mágneses Északi-sarkhoz viszonyítva

kényelmessé válik.



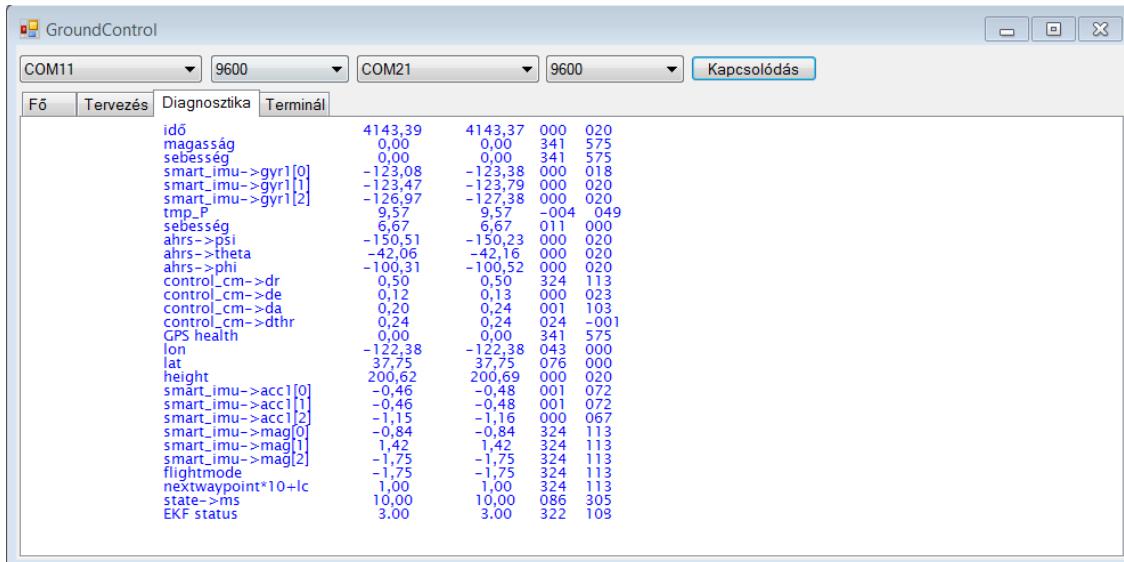
**3.3. ábra.** *GroundControl tervező felülete*



**3.4. ábra.** *Sikeres feltöltés tájékoztató ablaka*

### 3.2.3. Diagnosztikai képernyő

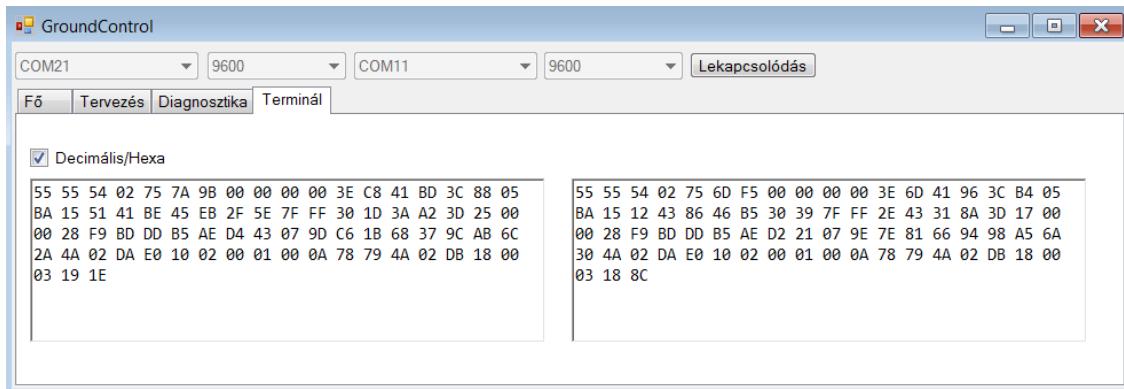
A diagnosztikai képernyőn a portokon érkező adatok és hozzájuk tatozó hibaszámláló láttható. Ha az egyik érték hibaszámlálója elér egy kritikus szintet, akkor pirossá válik és egy felugró ablakon figyelmezteti az operátort, hogy valamilyen hiba történt. Ő eldöntheti, hogy ez csak az egyik műszer meghibásodása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



3.5. ábra

### 3.2.4. Terminál képernyő

A fogadott csomagokat a terminál képernyő (3.6.) jeleníti meg. Ha az előző képernyőkön nem jelennének meg adatok, akkor itt megbizonyosodhat a felhasználó, hogy a kapcsolat egyáltalán működik-e. Ez a felület biztosítja egy csomag legalacsonyabb szintű kijelzését, mivel csak a fogadott csomag bájtjait írja ki választhatóan hexadecimális vagy decimális formában. Ha itt nem látható semmi, érdemes a csatlakoztatott portot vagy a jelsebességet megváltoztatni.



3.6. ábra. *GroundControl* terminál képernyő

### 3.3. Fejlesztés menete

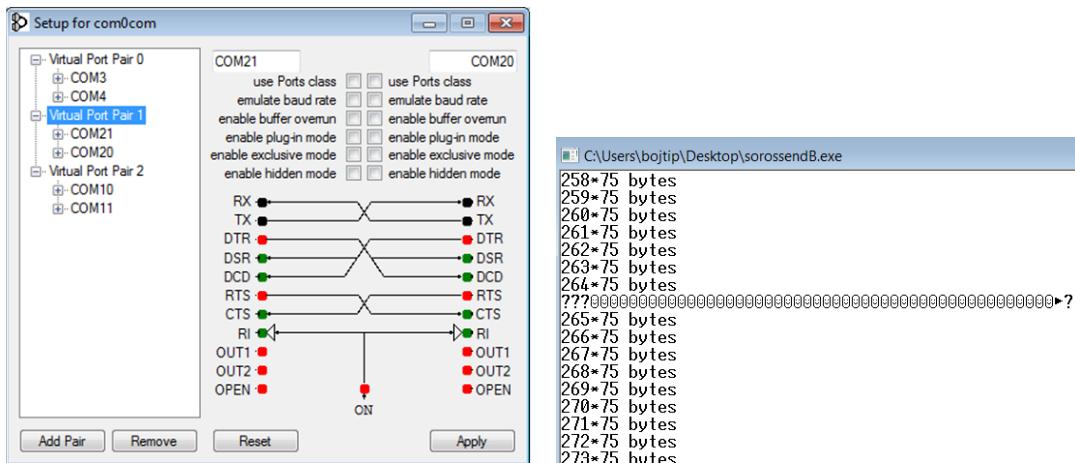
Az önálló, otthoni fejlesztést megkönnyítendő, a repülőgép HIL (Hardware In the Loop)<sup>2</sup> szimulációjából nyert log fájlokat használtam fel. Ennek segítségével nem volt szükséges ez idő alatt a repülő közelében lennem, a program működését elég volt csak a végleges stádiumban kipróbálni a repülőgép valódi adataival.

<sup>2</sup>olyan szimuláció, ami a tesztelendő beágyazott rendszernek olyan környezetet biztosít, mintha az valódi érzékelők adatait fogadná és valódi aktuátorokat irányítana

A kapott log fájlok felhasználásához először is biztosítani kellett, hogy a GroundControl soros porton keresztül ugyanúgy fogadhassa a csomagokat, mintha azt a repülőgép modemei küldenék. Ehhez készítettem egy-egy programot, melyek ezeket a fájlokat beolvassák és két kiválasztott soros portra 500 ms-onként egy-egy csomagot küldenek belőlük.

A repülőgépet helyettesítő programok és a GroundControl közti kapcsolatot egy emulátor programmal hoztam létre. Az emulátor program [25] képes null-modemként viselkedni, melynek lényege, hogy szoftveres úton biztosít két soros port között összeköttetést. Létrehoztam egy COM20-COM21 és egy COM10-COM11 port párosítást (3.7. ábra), melyekből az egyik program a COM20-ra, míg a másik a COM10-re csatlakozik. A GroundControl a párok másik portjait használja a kommunikációra.

Mivel szükséges a kétirányú kapcsolat kiépítése, így a log fájlokat feldolgozó programokba beépítettem egy beérkező üzeneteket feldolgozó függvény is. Ez kiírja a program konzolos felületére a beérkezett csomagot, így ellenőrizhető, hogy valóban működik-e a GroundControl szemszögeből az adatküldés. A 3.8. ábrán egy ilyen csomag látható a 264. és 265. küldött csomag között.



3.7. ábra. Com0com nullmodem program

3.8. ábra. Log fájlt feldolgozó program

A kapott log fájlok szerkezetét egy XVI32 nevű programmal tanulmányoztam. Egy csomag felépítését és méretét ezzel ellenőriztem. A 3.9. ábrán 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.



## 4. fejezet

# Összefoglalás

Szakdolgozatom keretében bemutattam a piacon lévő földi irányítóegységek grafikus felületeit. Melyek összehasonlításából levont követeztetéseket felhasználtam a tervezés fejezetben. Megterveztem, hogy a specifikáció megvalósításához milyen feladatok megoldása szükséges, melyet a megvalósítás fejezetben részleteztek. 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. A párhuzamosan, aszinkron érkező jelek közti eltérést a program detektálja és értesítést ad a kezelő személyzetnek, ha ez elér egy kritikus szintet. Emellett megoldottam az útvonal feltöltésének a lehetőségét is, így kényelmesen kijelölhető a repülőgép által bejárandó 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://hu.wikipedia.org/wiki/CAN\\_bus](http://hu.wikipedia.org/wiki/CAN_bus), 2013. december 12., 11:00
- [3] <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868>, 2013. október 29., 14:00
- [4] <http://www.makershed.com/v/vspfiles/assets/images/122-32450-xbeetutorial-v1.0.1.pdf>, 2013. december 12., 11:00
- [5] <http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>, 2013. november 29., 15:00
- [6] [http://www.digi.com/pdf/wp\\_zigbee.pdf](http://www.digi.com/pdf/wp_zigbee.pdf), 2013. november 29., 14:00
- [7] [http://hu.wikipedia.org/wiki/OSI\\_modell](http://hu.wikipedia.org/wiki/OSI_modell), 2013. december 12., 11:00
- [8] <http://www.sensor-networks.org/?page=0823123150>, 2013. december 02., 22:00
- [9] <http://www.uvisionuav.com/portfolio/gcs/>, 2013. november 24., 15:00
- [10] [http://personal.us.es/imaza/papers/journals/maza\\_jint10\\_multimodal/maza\\_jint10\\_multimodal](http://personal.us.es/imaza/papers/journals/maza_jint10_multimodal/maza_jint10_multimodal), 2013. november 24., 15:00
- [11] <https://code.google.com/p/ardupilot-mega/wiki/Mission>, 2013. november 24., 15:00
- [12] <http://arduino.cc/>, 2013. december 11., 23:00
- [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](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/hh425099.aspx>, 2013. december 5, 12:00
- [21] <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>, 2013. december 5, 12:00
- [22] <https://www.aut.bme.hu/Course/VIAUA218>, 2013. december 5, 12:00
- [23] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms533798.aspx>, 2013. december 3, 12:00
- [24] <http://greatmaps.codeplex.com/>, 2013. december 8, 11:00
- [25] <http://com0com.sourceforge.net/>, 2013. május 4, 10:00

# Függelék

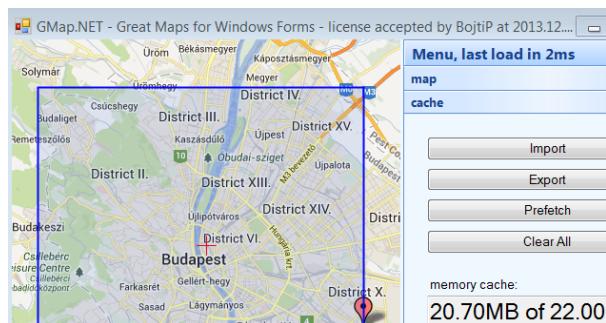
## F.1. Program elindítása

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

A program futtatásához Windows XP/Vista/7 és .NET 4.5-s verzió szükséges. Telepítője a mellékleten szerepel *dotNetFx45.exe*<sup>1</sup> 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

<sup>1</sup>Internet elérés szükséges