



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 7.

# Tartalomjegyzék

<b>Kivonat</b>	<b>4</b>
<b>Abstract</b>	<b>5</b>
<b>1. Bevezetés</b>	<b>6</b>
1.1. Motiváció . . . . .	6
1.2. Előzmények . . . . .	7
1.2.1. Repülőgép felépítése . . . . .	7
1.2.2. Architektúra . . . . .	8
1.2.3. Kommunikáció . . . . .	10
1.3. Saját feladatom részletezése . . . . .	12
1.3.1. Mi a földi irányító állomás feladata? . . . . .	12
1.3.2. Hibatűrő kommunikáció kialakítása . . . . .	12
1.3.3. Grafikus felhasználói felület . . . . .	13
1.4. Földi állomások bemutatása . . . . .	13
1.4.1. ArduPilot . . . . .	13
1.4.2. Paparazzi . . . . .	15
1.4.3. MicroPilot Horizon . . . . .	17
1.4.4. OpenPilot . . . . .	18
1.4.5. QGroundControl . . . . .	19
1.4.6. HappyKillmore . . . . .	20
1.4.7. Összehasonlítás . . . . .	21
<b>2. Tervezés</b>	<b>22</b>
2.1. Kommunikáció . . . . .	22
2.2. Adatok fogadása . . . . .	22
2.2.1. Protokoll . . . . .	22
2.3. Adatok küldése . . . . .	24
2.3.1. Protokoll . . . . .	24
2.4. Grafikus felület . . . . .	25
2.4.1. Főképernyő . . . . .	26
2.4.2. Tervezés képernyő . . . . .	26
2.4.3. Diagnosztikai képernyő . . . . .	26
2.4.4. Terminál képernyő . . . . .	26

2.5. Használt technológiák bemutatása . . . . .	27
2.5.1. .NET . . . . .	27
<b>3. Megvalósítás</b>	<b>29</b>
3.1. Program felépítése . . . . .	29
3.1.1. Kapcsolódás megvalósítása . . . . .	30
3.1.2. Redundáns adatok feldolgozása . . . . .	30
3.1.3. Redundáns adatok hibadetektációja . . . . .	30
3.2. Megjelenítés . . . . .	31
3.2.1. Főképernyő . . . . .	31
3.2.2. Tervezés képernyő . . . . .	33
3.3. Diagnosztikai képernyő . . . . .	33
3.4. Terminál képernyő . . . . .	34
3.5. Fejlesztés menete . . . . .	34
<b>4. Értékelés</b>	<b>36</b>
<b>5. Összefoglalás</b>	<b>37</b>
<b>Függelék</b>	<b>40</b>
F.1. Függelék1 . . . . .	40

## 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 7.

---

*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 hardware elemekkel védekezik. Szakdolgozatom keretében ennek a repülőnek a földi állomását dolgoztam ki, mely a redundánsan küldött rádiójelek feldolgozására és megfelelő megjelenítésre használandó. A földi személyzet mozgó térkép alapú vizualizáció láthatja az aktív útvonalpontokat és a gép útvonalát, a diagnosztikai adatokat és esetleges hibákat egy másik nézetben áttekintheti. Lehetőség nyílik repülési terv meghatározására és feltöltésére a repülőre, melynek fordulópontjait követi.

# **Abstract**

Nowadays..

# 1. fejezet

## Bevezetés

### 1.1. Motiváció

Az MTA SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumában folyó kutatások eredményének demonstrálására szükség volt egy olyan eszköz megalkotására, mely a kifejlesztett szabályozó algoritmusok működését látványosan mutatja be. Így esett a választás egy repülőgépre, aminek a levegőben tartása sem triviális, szükség van a kormányszervek harmonikus mozgatására, a tolóerő szabályzására. Ezen túlmenően, ha komplikáltabb feladatok végrehajtására van szükség, pl. fordulópontokat követve feltérképezni egy ismeretlen területet, már számolni kell a széllel, mely eltérítheti az útvonaláról, felszálló légáramlatokkal, melyekre gyorsan kell reagálnia. Mivel egy teljesen felszerelt repülő összeállítása, felprogramozása nagy szakértelmet, sok időt, energiát igényel és nem utolsó sorban anyagi ráfordítást, 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, amelyben cél minden egyes repülőgép alrendszer meghibásodásának diagnosztizálása, 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 megoldás 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 értelmezheték, és az esetleges meghibásodásokról is időben értesüljenek.

## **1.2. Előzmények**

A pilóta nélküli légijármű, azaz UAV (Unmanned 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 funkcionálisága 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 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 pozícióját, diagnosztikai adatait. Ezt a feladatot látja el az ún. földi irányító állomás, azaz GCS (Ground Control Station). Diplomatervem célja egy meglevő pilóta nélküli légijármű földi irányító állomásának megtervezése és elkészítése, a megbízható működés követelményeinek figyelembe vételével.

### **1.2.1. Repülőgép felépítése**

A jelenlegi repülő egy saját építésű 3.2 m fesztaávolságú modell, melybe diagnosztikai és hibadetekciós célokra egyedi tervezésű komponensek kerültek. A kommerciális célokra szánt szervó motorok nem szolgálnak elég információval a kitérésükről, fogyasztásukról, ezért ezek méréséről gondoskodni kellett.



**1.1. ábra**

### **1.2.2. Architektúra**

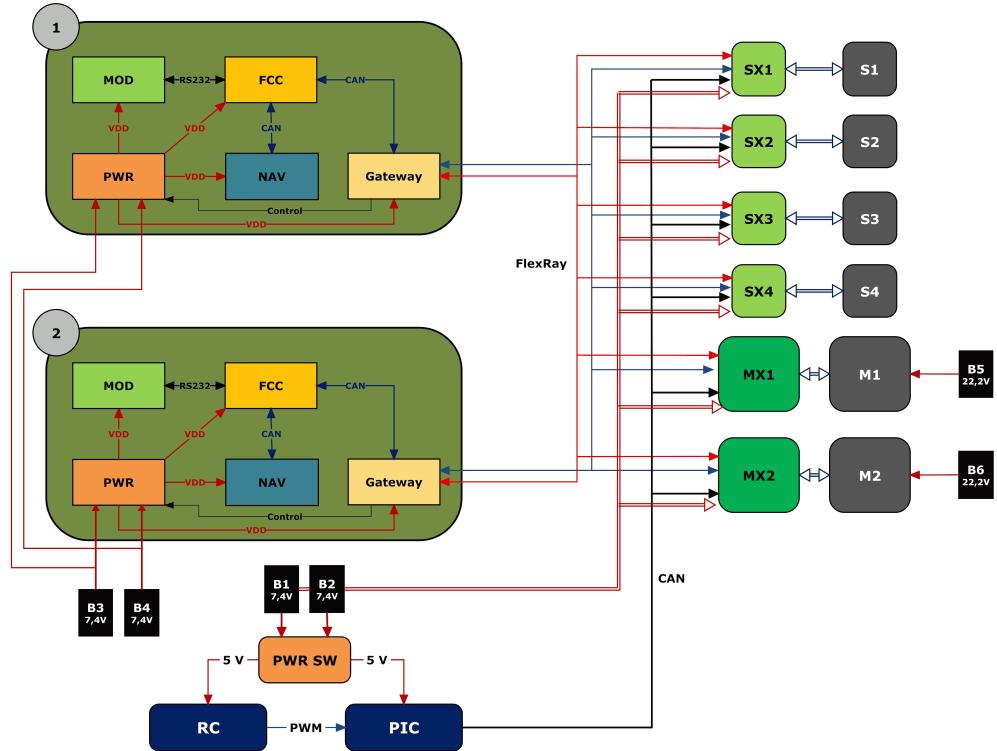
Elsődleges szempont, hogy egy komponens meghibásodása ne okozza a gép vesztét, tehát a rendszerben ne maradjon SPOF<sup>1</sup>. Ezt redundanciával érhetjük el, mely során a repülőgépen duplikáljuk az elemeket, melyek nélkülözhetetlenek a levegőben maradásához:

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

A repülő méretéből adódóan térbeli szeparációval nem lehetséges a megbízhatóság növelése, mint pl. harci repülőgépek fedélzeti számítógépeinél, melyek a találat kockázata miatt elszórva, akár 4–5x-ve vannak.

---

<sup>1</sup>Single Point Of Failure, olyan meghibásodás, mely ha bekövetkezik, az egész rendszer leállásához vezet



1.2. ábra

1.2. ábrán látható a kialakított architektúra. Elemei:

- **FCC** Flight Controll Computer, repülőgép irányítása
- **NAV** GPS, nyomásmérő, orientációmérő
- **MOD** modem, kommunikáció
- **Gateway** CAN-FlexRay átváltást biztosít
- **PWR** feszültségválasztó
- **M1, M2** motorok
- **S1, S2, S3, S4** szervók
- **Sx, Mx** szabályzóelektronika

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 (NAV) eszköz szolgáltatja a GPS-ból érkező magasság és pozíció adatokat, az IMU<sup>2</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 buszon keresztül zajlik. Az ábrán látható S és M-mel jelölt elemek rendre a szervó motorok és a meghajtásért felelős

<sup>2</sup>Inertial Measurement Unit, orientáció- és gyorsulásmérő berendezés

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 ezek át vannak alakítva, hogy a FlexRay hálózatra illesztésért felelős szabályozóelektronikájuk (Sx, Mx) megfelelően működhessen. Ezekre bármilyen szabályzóalgoritmus írható, hibadiagnosztikai célokra fault detection filtert vagy Kármán szűrő alkalmazható. A szervók mágneses enkódere a kitérésről, a motorok elektronikája a fogyasztásról ad információt. A dolgozat szempontjából a legérdekesebb egység a modem (MOD) mely az FCC-vel soros porton kommunikál. 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 mindenleg a legnagyobb feszültséggel operálót választani vagy egyiket lemeríteni bizonyos százalékig és ezután váltani. A B5 és B6 a motorokat hivatottak kiszolgálni, ezek a legnagyobb fogyasztásúak, így ezek kapacitása a legnagyobb. B1 és B2 biztosítja az aktuátoroknak, az hozzájuk tartozó vezérlőknek az áramforrást. Továbbá, mivel a repülőgép mindenleg csak távirányítással tud felszállni, így a távvezérlő egység (RC) és a PWM-CAN átalakításért felelős PIC is ezt a 2 akkumuláltot használja. A PIC közvetlenül az szabályzó elektronikákhoz csatlakozik CAN interface-en, ez a legközvetlenebb módja a kézi irányításnak.

### 1.2.3. Kommunikáció

A fedélzeti MOD egység egy XBee-PRO 868 típusú modem [4], mely alacsony fogyasztása és nagy hatótávolsága miatt ideális egy ilyen környezetbe. Vevő oldalon ugyanilyen modem található duplikáltan, mely szintén soros porton küldi a földi állomásnak a vett jelet, a modemelek között lévő vezeték nélküli protokoll: 802.15.4, melyet a 1.2.3 fejezet foglalkozik.

#### Vezeték nélküli modem

A kiválasztott modem kétféleképpen képes kommunikálni:

- API csomagküldés
- AT transzparens

**API** módban egy eszköz több eszköztől tud csomagokat venni, ha egy csomag megérkezik a küldőtől a fogadóig, egy ACK<sup>3</sup> üzenettel válaszol, ha ezt nem kapja meg, a csomagot újraküldi, lehetőség van broadcast üzenetek küldésére is, melyet minden eszköz megkap. Egy csomag többek közt tartalmazza a küldő és a fogadó címét, az adatot és adatintegritás céljából checksum mezőjében összesítve csomag tartalmát.

**AT** mód egy vezeték nélküli kapcsolatot jelent 2 sorosport közt. A modem a soros portján bejövő adatokat rádiójelekké alakítja, melyet a virtuális kapcsolat végpontja fogad és visszaalakítja soros portra. Ez pont-pont kommunikációra hivatott, egyéb topológia nem támogatott.

---

<sup>3</sup>Acknowledgement, nyugtázó üzenet, melyet a fogadó küld a feladónak egy csomag sikeres vétele esetén

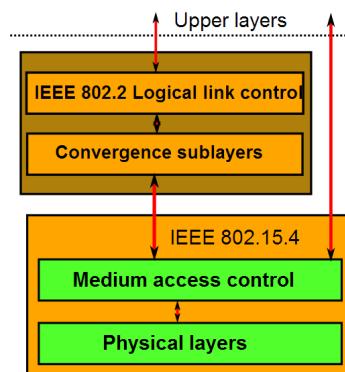


**1.3. ábra**

### IEEE 802.15.4 vezeték nélküli protokoll

Az IEEE<sup>4</sup>-nek egy csoporthoz a 802.15 mely a WPAN<sup>5</sup> hálózatok szabványosításával foglalkozik, 7 alcsoporthoz közül a 802.15.4 [6] a Low Rate WPAN nevet viseli. Ez a szabvány a kis fogyasztású, olcsó és alacsony sávszélességű vezeték nélküli kommunikációval foglalkozik. Az OSI modell első (fizikai) és második (adatkapcsolati) rétegét valósítja meg, erre építkezik a ZigBee [5] cég által specifikált protokoll verem.

A fizikai réteg feladata fizikai összeköttetést teremteni a hardware-rel, specifikálja, 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<sup>6</sup> tartományban helyezkedik el. Továbbá átvitási biztosítást biztosít a felette lévő adatkapcsolati rétegeknek, melynek feladata a hibamentes átvitel biztosítása 2 pont között hibajavítással/jelzéssel és forgalomszabályzással. A biteket keretekbe ágyazva küldi a felsőbb rétegeknek, maximális mérete 127 byte melynek formátuma a IEEE 802.15.4-2011-ben van specifikálva, 16 bites CRC ellenőrző kóddal zárul.



**1.4. ábra**

2 fajta topológia kialakítása lehetséges, egyik a pont-pont másik a csillag. Mindegyik eszköz egy egyéni 64 bites azonosítóval rendelkezik.

A Zigbee 2 újabb réteget helyez az eddigiek fölé, a hálózati és az alkalmazásit, melybe

<sup>4</sup>Institute of Electrical and Electronics Engineers

<sup>5</sup>Wireless Personal Area Network

<sup>6</sup>Industrial, Scientific and Medical (ISM) sávok, melyek szabadon használhatóak ipari, tudományos és orvosi területen

belekerült a ZDOs<sup>7</sup>, mely az eszközök csatlakozásáért, felderítéséért és biztonságáért felelős.

Alacsony fogyasztása abban rejlik, hogy alvó állapotból 30 ms alatt aktívrá tud váltani, így gyakran tud alacsony fogyasztású állapotban lenni, jelentős késleltetés nélkül. Nagy hatótávolságot a 868 MHz-es verzióval a BPSK [7] (Binary Phase-Shift Keying) és Direct Sequence Spread Spectrum (DSSS) technológiák keresztezésével érték el, mely nagy jel-zaj viszonyt biztosít. Az BPSK lényege, a 0-1 és 1-0 átmenetet a vivőfrekvencia 180 fokos fordításával reprezentálja, mely jelentős zajt képes elviselni, a DSSS egy bitet 4 biten reprezentál, igaz ez csökkenti a sávszélességet, de robosztusabbá teszi az információt. A megbízhatóságot Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) segítségével érték el, mielőtt egy állomás adna, megnézi hogy a csatornán zajlik-e kommunikáció, ha igen akkor különböző stratégiák segítségével vár az üzenet küldésének újból próbálásával.

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

### **1.3. Saját feladatom részletezése**

Feladatom közé tartozik 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 irányítja, monitorozhatja a repülést. Általában több [2] funkciót lát el:

- Küldetés tervezés: útvonal meghatározása, illetve a célpontok kijelölése
- Küldetés végrehajtás: távirányítással vezetve az operátor végrehajtja a feladatot
- Adatok megjelenítése: megfelelő módon kijelezni a repülőgép állapotát, esetleges hibáit

#### **1.3.2. Hibatűrő kommunikáció kialakítása**

##### **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. Figyelembe kell venni, hogy feladataim közé tartozik az útvonalpontok feltöltésének lehetségeit

##### **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 számolni kell.

---

<sup>7</sup>ZigBee Device Objects

## Biztonságos protokoll

2 irányú 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 megjelenítés felhasználóbarát legyen, több nézeti oldal szükséges:

- 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 állomások bemutatása

Számos megoldás született földi állomások GUI<sup>8</sup>-jainak kialakítására. Elsődleges követelmény, hogy az operátor mindenkor minden információt láthassa, ehhez szoftverengelőkkel kell megtervezni a műszerek. Kísérleteket folytattak, milyen elrendezésben, hány képernyőn érdemes megjeleníteni az adatokat, úgy, hogy az még ne terhelje túl az operátort [3]. Bebizonyosodott, hogy érdemes több módon jelzést adni, így pl. nagy prioritású eseménynél a figyelmeztető ablak megjelenését hanghatás is kíséri. Alábbiakban összehasonlításra kerülnek a piacra lévő megjelenítési felületek, megoldások.

### 1.4.1. ArduPilot

Az Ardupilot projekt [8] létrejöttének oka, hogy otthoni körülmények között, nem ipari alkatrészekből bárki összeállíthat a robotrepülőgép, mely az előre betáplált utasításokat végrehajtja. Központi eleme az Arduino cég által készített ATMEL mikroprocesszorra épített platform, mely felhasználóbarátabbá teszi a mikroprocesszor programozását. Magas szintű utasításokkal segíti az eszközzel való barátosságot, nincs szükség assembly szintű tudásra. Erre a platformra hozták létre az ArduPilot programot, képes vezérelni többféle autonóm járművet:

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

---

<sup>8</sup>Graphics User Interface, grafikus megjelenítés

- Arduover: 4 kerekű autó

Sikerességének fő oka, a nyílt forráskód, Arduino panel alacsony ára ( 5 ezer Ft) és a projekt mögött álló lelkes közösséggel.

Ennek a közösségnak köszönhetően született a Mission Planner nevű szoftver, mely teljes körű támogatást nyújt a csatlakoztatott járműveknek.

## Főképernyő

Több nézet segítségével könnyen átlátható a funkcionalitása, repülés szempontjából a főképernyőn 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 különbségből számítható). A legnagyobb területet a térkép foglalja el, melyen az aktuális irány, lerepült útvonal, fordulópontok láthatóak. Megfigyelhető, hogy ez kapja arányaiban a legnagyobb területet.



1.5. ábra

Ez a nézet akkor jöhét jól, mikor olyan meghibásodás történik, melyre nincs felkészítve az irányítóegység és szükséges lehet a kézi üzemmódra váltás. Előfordulhat ilyen esetben, hogy nincs vizuális rálátás az operátor és a repülőgép között, ekkor csak az itt látható műszerek és térkép alapján szükséges irányítania. Szerencsére ez az avionikában már bizonyított, hogy műszerek segítségével, „vakon” is lehetséges repülni, itt azonban ez az eset csak pár percig szükséges.

## Tervező és útvonalfeltöltő

Másik nézet lehetőséget biztosít útvonalpontok kijelölésére, az útvonalpontokhoz egy listából kiválasztható, hogy az adott pont start-, forduló-, végpont és egyéb lehetőségek. Az útvonalpontok pozícióját egérrel módosíthatjuk, törlhetjük. Bal felső sarokban a kijelölt tervnek hosszát láthatjuk, ez segítséget nyújt, nehogy túlhaladjuk a rádiókapcsolat és a repülő 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										
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		

1.6. ábra

Mindig látható a csatlakozás gomb, melynél a soros port és a jelszabályzás beállítása után a csatlakozás gombbal csatlakozik a program a portra.

## Összegzés

Felhasználói szempontból barátságos felületet biztosít a különböző nézetekkel, gombok átgondolt elhelyezésével. Nem a program hibája, de nincs felkészítve párhuzamos csatornák kezelésére, ez az ArduPilot projekt egyszerűségének köszönhető, mivel nem használ redundanciát. Mivel a repülő, melyhez a programot készítem, nagy megbízhatóságú, így szükséges megoldani a 2 port kezelését és az azokon érkező adatok feldolgozását. Jó ötlet a csatlakozás gomb mindenkor látható elhelyezése, a főképernyón térképen ábrázolni a repülő helyzetét, illetve a műhorizont. Azonban hibadiagnosztikai kijelzés nincs megoldva, melynek szintén fontos a megvalósítása.

### 1.4.2. Paparazzi

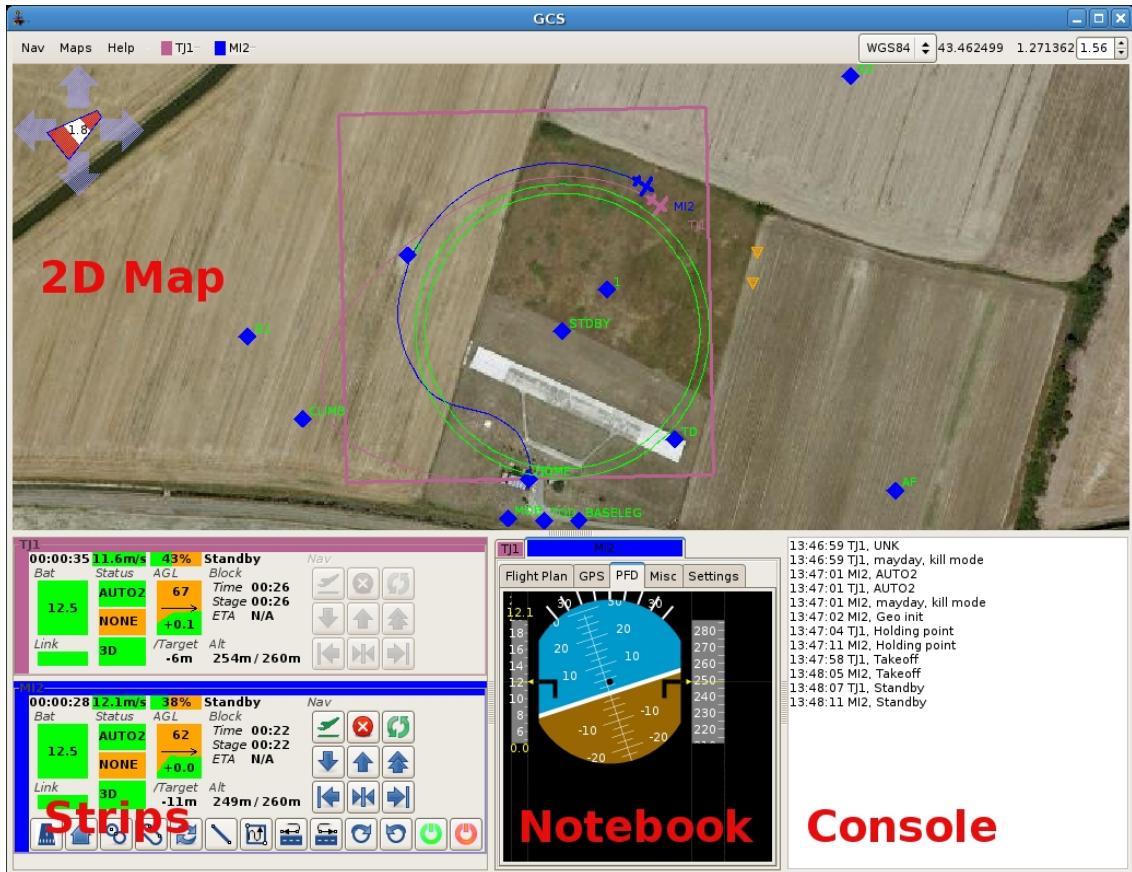
Hasonlóan mint az ArduPilot, ez a projekt [9] is a könnyen, otthon összeszerelhető repülő megépítése jegyében született. Nem csak egy nyílt forráskódú robotpilótát ad, hanem komplett „csináld magad” készletet, melyben feszültségszabályzótól kezdve a GPS vevőig minden megkaphatunk. Továbbá a hozzá készült földi állomáshoz antennát, modemet és programot is nyújt.

A program a következő funkciókkal rendelkezik:

- több platform támogatása (fix- és forgószárny)
- több jármű egyidejű kezelése
- Google Maps, OpenStreetMaps, Microsoft Maps térképes megjelenítés
- küldetés tervezés

- mozgatható iránypontok
- hangjelzés

## Főképernyő



1.7. ábra

1.7. ábrán látható képernyő nagy százalékát a térkép foglalja el, melyen a lerepült útvonal és a fordulópontok láthatóak. A repülő aktuális helyzete mellett a neve, sebessége és repülési magassága is kijelezve van. Bal oldalt egy információs sávban a következő fontosabb adatok láthatóak: akkumulátor töltöttsége, sebesség, tolóerő, magasság, illetve utasítások: fel-, leszállás, megfigyelés indítása. Jobb felső sarokban a kurzor térképen lévő koordinátája kerül megjelenítésre.

A térképet billentyűzettel és egérrel lehet mozgatni, nagyítani. A fordulópontokat is ezen a felületen lehet szerkeszteni. A módosítások egy megerősítő üzenet jóváhagyása után töltődnek fel a repülőre, melyre az egy megerősítő válasszal reagál. Új pontot csak felszállás előtt lehetséges feltölteni.

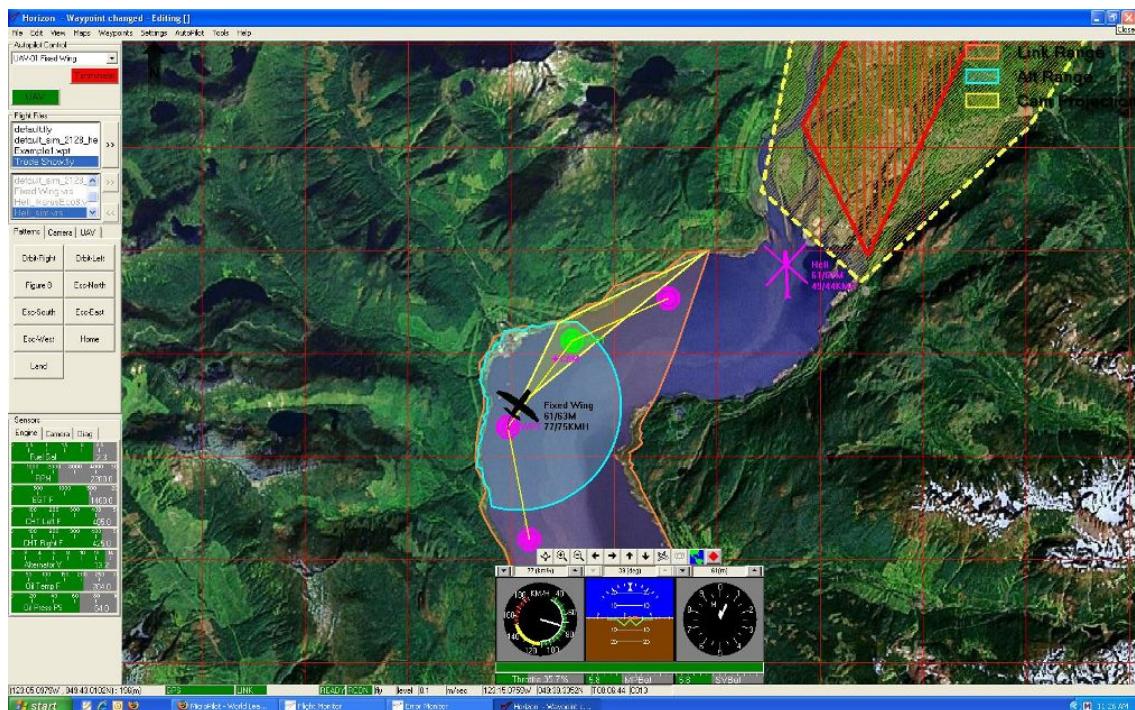
Középen alul található egy több füllel ellátott rész, melynél kiválasztható, hogy építeni egy műhorizontot, a GPS által vett adatokat, az útvonaltervet vagy a beállításokat szeretnénk-e látni.

## Összegzés

Egy képernyő szolgál a navigációra és az útvonal-kijelölésre, mely szerintem nem a legjobb megoldás, mivel nincs elválasztva ez a két funkcionálitás. Összességében ez is egy nagyon jól használható program, mely teljesen kiszolgálja az UAV-kat melyhez készítették.

### 1.4.3. MicroPilot Horizon

„Az 1995 óta működő kanadai [10] MicroPilot 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 (RCON) 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 szervo állást őrzi meg, vagy egy előre programozott legbiztonságosabb földet érést igérő beállításra ugrik.” [11] A hozzá készített földi irányító egység a MicroPilot Horizon nevet viseli.



1.8. 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 átlátszóan jelennek meg, így nem kell másik képernyőre tekintenie a kezelőnek. Több eszközön tud egyszerre csatlakozni és kiszolgálni, mindegyikhez külön név rendelhető. Az összes csatlakoztatott jármű között szinkronizálva vannak az útvonalpontok. Számunkra érdekes lehet, hogy a felmerülő hibákhoz lehetőség van különböző prioritási szintek meghatározására, hangjelzés hozzárendelésére. Így egy bizonyos szint alatti hibák nem terelik el az operátor figyelmét.

## Tervező képernyő

Repülési terv könnyen, kattintással összeállítható és módosítható, minden repülés előtt és közben. POI<sup>9</sup> gombnyomásra lerakható a térképen, ha esetleg valami olyat lát az operátor, amit érdemes utána ismét megvizsgálni. Az útvonal hosszát folyamatosan kijelzi tervezés üzemmódban.

## Összegzés

Ez a program elsősorban távirányításos üzemmód támogatására készült, mely során az operátor irányítja a gépet és ehhez a legtöbb információt szolgáltatja. Így kulcsfontosságú a videókép átvitel támogatása és a legfelhasználóbarátabb megjelenítés.

### 1.4.4. OpenPilot

A projekt célja [12], hogy nyílt forráskódú, magas minőségű robot pilótát hozzanak létre autonóm repülőkhöz. 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 2-től 8 rotorig.

A hozzá készült GCS több platformon működik: Windows, Mac OS, Linux, tervben van az Android támogatás is. Segítségével tölthető fel a panelra a fedélzeti program, illetve interface-t biztosít a repülés alatt.

## Főképernyő

Bal oldalon műhorizont látható, alatta az aktuális 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 helyett egy mért érték grafikus megjelenítésére kicserélhető. Ezek .xml fájlba elmenthetőek és visszatölthetőek, különböző konfiguráció igényehez igazodva.

---

<sup>9</sup>Point Of Interest, érdekes hely mely GPS koordinátával megjelölhető



**1.9. ábra**

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

#### 1.4.5. QGroundControl

MAVLink protokollt használ a kommunikációra, a repülőgép a programon keresztül vezethető, ha átkapcsolás szükséges. A MAVLink kifejezetten GCS–UAV összeköttetésére lett kifejlesztve [13].

#### Főképernyő

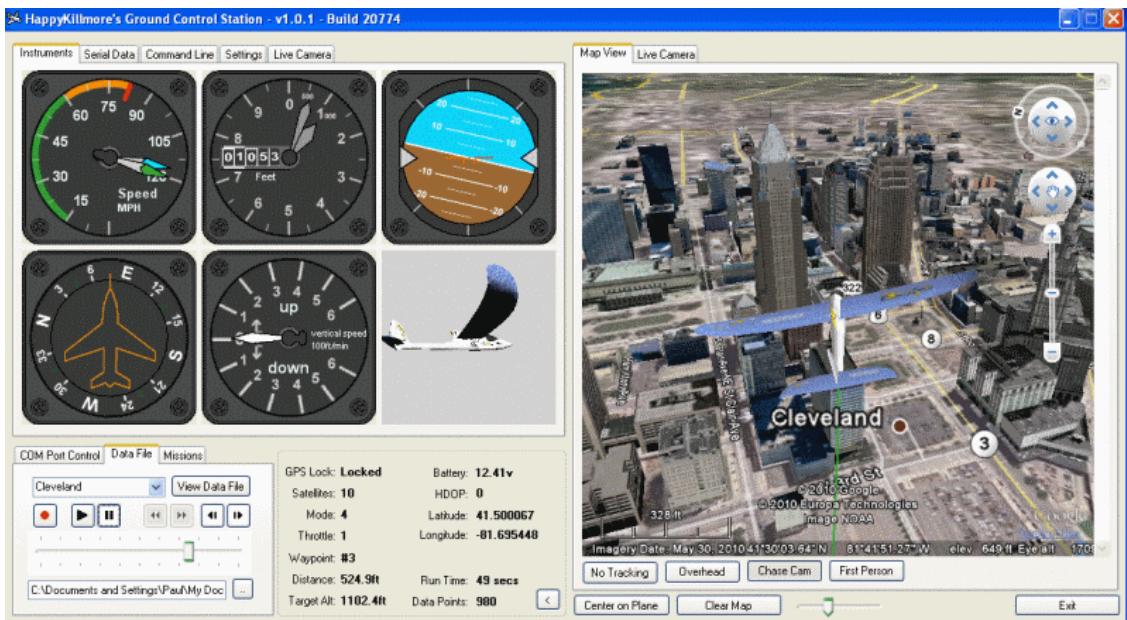
Érdekesség, hogy 3D-ben is meg tudja jeleníteni a repülő aktuális pozícióját. A protokollnak köszönhetően 255 jármű egyidejű kezelésére alkalmas. Útvonalpontok menet közben is változtathatóak.



1.10. ábra

#### 1.4.6. HappyKillmore

ArduPilot-hoz készült nyílt forráskódú GCS [14], 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.



1.11. 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 megjelenést 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 eszközhöz való csatlakozás megoldása, ám 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.

Program	Ardupilot	Paparazzi	MicroPilot	QGround Control	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**

### **2.1. Kommunikáció**

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 tároló kell, mely a legutóbb küldötteket tárolja egy FIFO listában.

#### **2.2.1. Protokoll**

Az adatokat a repülőgép 2 Hz-s frekvenciával küldi, ezek csomagokban érkeznek, melyeknek a felépítése: Egy UUT fejléc, adattagok, checksum, ha a ez nem egyezik a csomag összegével, akkor az a 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
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
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		0xFFFF	[-1..1]-re normálva
28			2/2				
29		uint16	1/2	control_cm->de		0xFFFF	[-1..1]-re normálva
30			2/2				
31		uint16	1/2	control_cm->da		0xFFFF	[-1..1]-re normálva
32			2/2				
33	normalizált gázkarállás	uint16	1/2	control_cm->dthr		0xFFFF	[0..1]-re normálva
34			2/2				
35	GPS egészség		uint16	1/2	TRUE/FALSE		
36			2/2				1 vagy 0
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	0xFFFF / 5,0
50			2/2				
51		uint16	1/2	smart_imu->acc1[1]	2,50	0xFFFF / 5,0	g
52			2/2				
53		uint16	1/2	smart_imu->acc1[2]	2,50	0xFFFF / 5,0	g
54			2/2				
55	mágneses térerősség	uint16	1/2	smart_imu->mag[0]	2,00	0xFFFF / 4	
56			2/2				
57		uint16	1/2	smart_imu->mag[1]	2,00	0xFFFF / 4	
58			2/2				
59		uint16	1/2	smart_imu->mag[2]	2,00	0xFFFF / 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	23		
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-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, 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? Láthatunk 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

---

<sup>1</sup>Bináris 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.

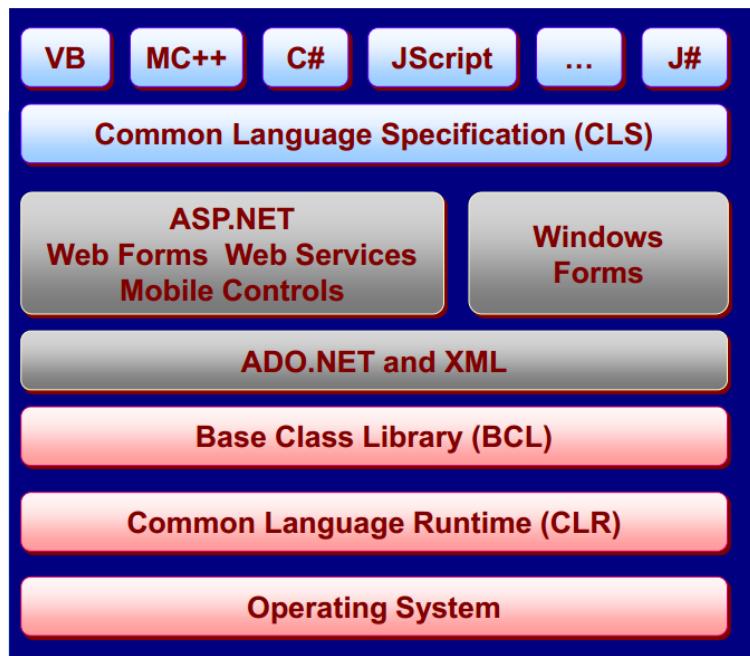
#### **2.4.4. Terminál képernyő**

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 [17] egy menedzselt végrehajtási környezetet biztosító keretrendszer, mely számos szolgáltatást nyújt a benne futtatott programoknak. Két fő részből áll: CLR<sup>2</sup>, mely a programok végrehajtásáért felelős és a .NET BLR<sup>3</sup>, mely tesztelt, újra felhasználható programkönyvtárakat tartalmaz.

#### Common Language Runtime

A menedzselt környezetet a CLR [18] 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

API<sup>4</sup>-k összesége, célja hogy megkönnyítse és gyorsítsa fejlesztés folyamatát. A feladatom megoldásához egyik legfontosabb osztály a *SerialPort*[15] osztály, mely megkönnyíti a soros port kezelését.

<sup>2</sup>Common Language Runtime (közös nyelvi futatókörnyezet)

<sup>3</sup>Base Class Library

<sup>4</sup>Application Programming Interface, 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 [19].

## **Windows Forms**

Grafikus megjelenítést biztosít az alkalmazásoknak, különböző elemek (beviteli mező, gomb, kép, választó lista) helyezhetők el rá. GDI<sup>5</sup> [16] 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 [19]. Más nyelvekkel összehasonlítva tisztább, mint a C++, nagy hasonlóságot mutat a Java nyelvvel.

---

<sup>5</sup>Graphic Device Interface

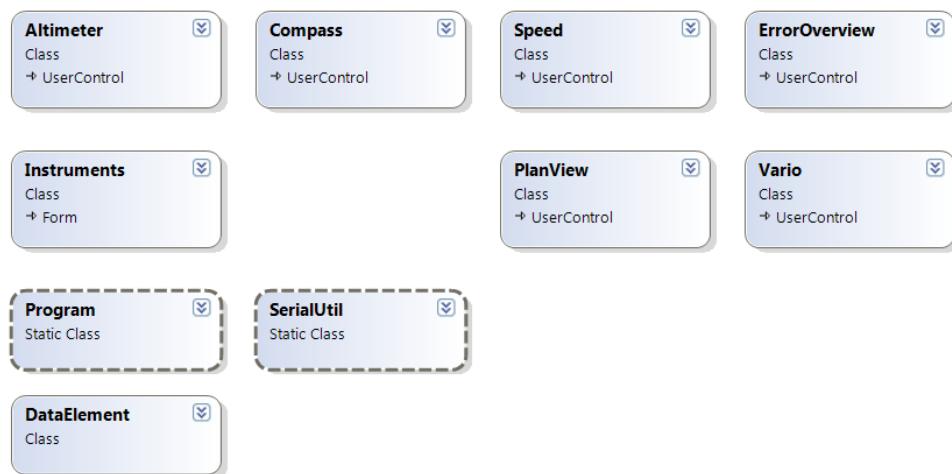
### 3. fejezet

## 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 3.5 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 műlhat, 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  $\epsilon^1$ -nál kisebb, akkor növelem a port hibaszámlálóját.

**Túl nagy ugrás:** TODO

**Kettő jel eltérése:** TODO

### 3.2. Megjelenítés

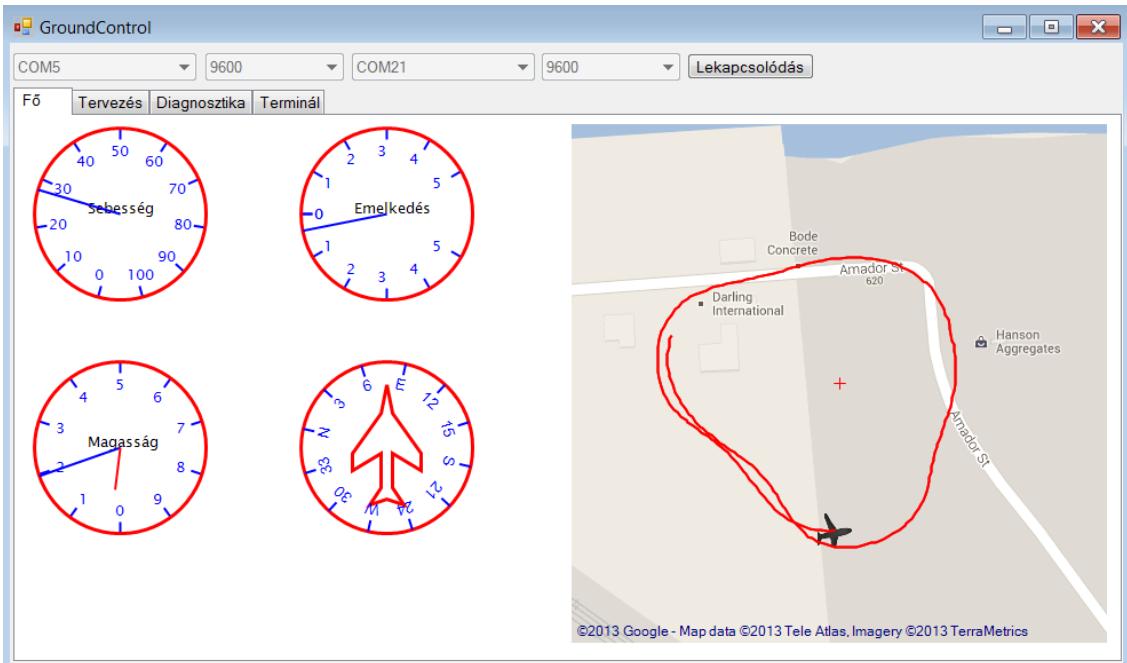
Bal oldalon a létrehozott *UserControl*-ok megjelenítése és elhelyezése a 3.2. ábrán látható. A terület nagyobb részét a térkép foglalja el, melyen a repülőgép aktuális pozíciója látható, illetve a múltban fogadott koordináták tört vonallal összekötve, mely a lerepült útvonalat ábrázolja. A program minimális 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 szemlélni 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. Elképzítettem a magasságmérő, emelkedésjelző, sebességmérő és iránytű *UserControll*-t melyek tervező nézetben „Drag and Drop” technológiával a megfelelő helyre húzható és könnyen beköthető.

---

<sup>1</sup>ez 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, ezt mi a *View* *Validate()* metódusával explicit is kiválthatjuk. GDI+ segítségével egy piros kört rajzolok

### Iránytű

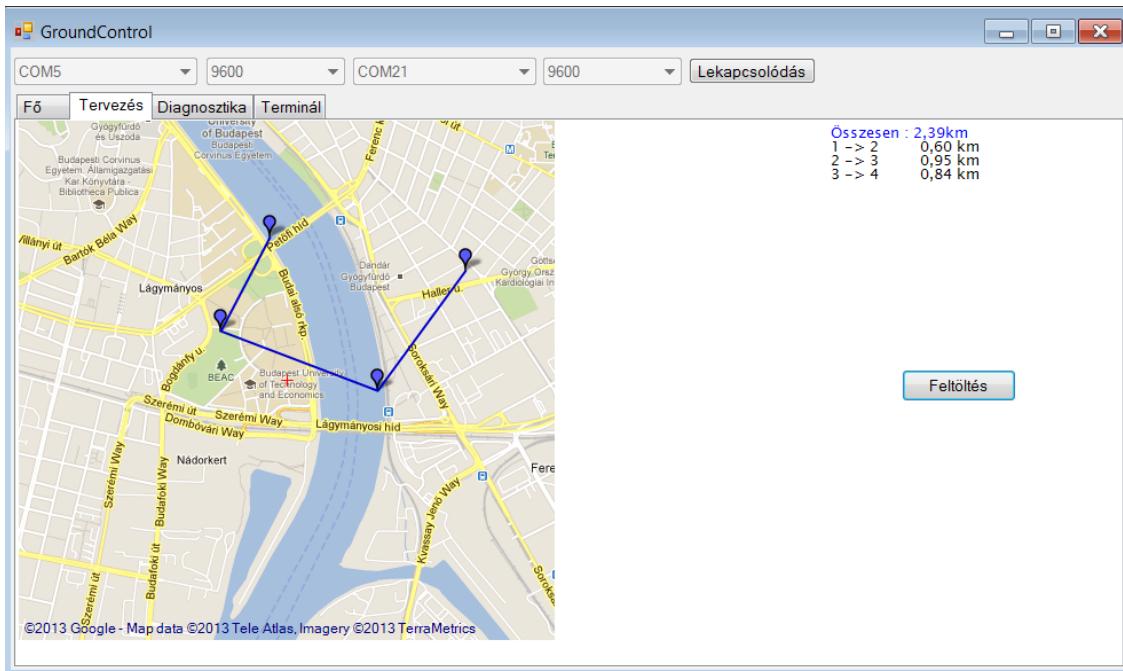
A iránytű felépítése hasonló , mint a sebességmérő műszernek, jelentős változás az égtájak és a fokok forgatását végző transzformáció. Először a kiírandó karaktert a grafikai koordináta-rendszer középpontjába tolom el, ekkor lehetséges az adott szöggel történő elforgatás, majd a megfelelő helyre eltolás. 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.

### 3.2.2. Tervezés képernyő

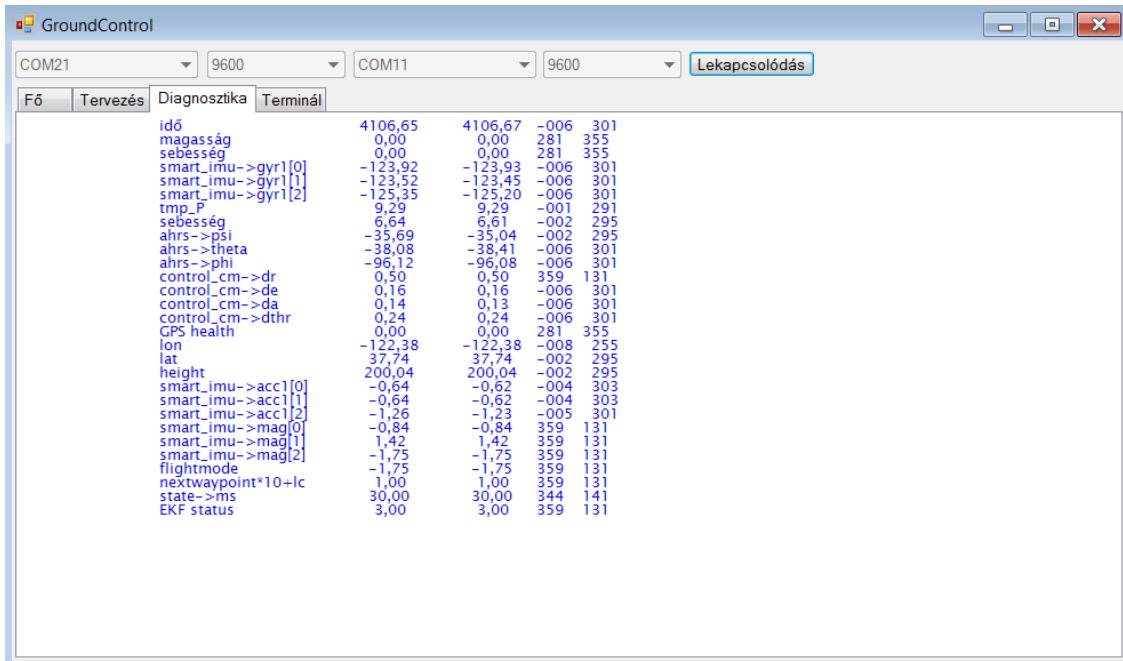
TODO



3.3. ábra

### 3.3. Diagnosztikai képernyő

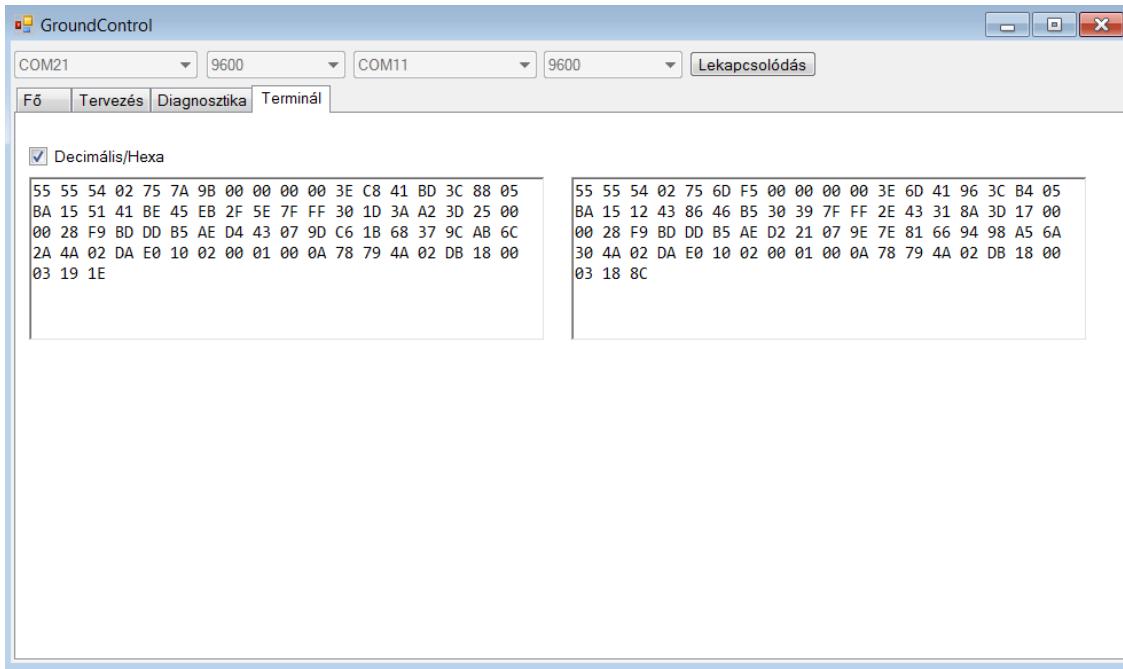
TODO



3.4. ábra

### 3.4. Terminál képernyő

TODO

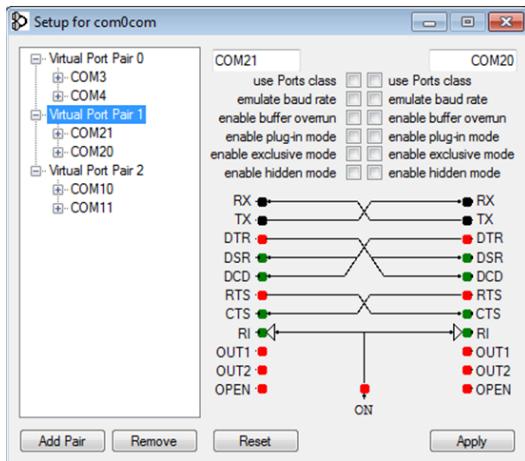


3.5. ábra

### 3.5. Fejlesztés menete

Az önálló, otthoni fejlesztést megkönnyítendő, a repülőgép HIL<sup>2</sup> 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 [20] 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.

<sup>2</sup>Hardware In the Loop, todo



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
```

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 iránypontok csomagjának helyességéről is.

XVI32 - loga.txt	
	File Edit Search Address Bookmarks Tools XVIscript Help
0 55 55 54 00 11 AB B0 3F F4 40 91 40 7C 3C 7E 41 F1	U U T < `` ° ? ó @ ' @   < ~ Á ñ
11 3C C2 3C 79 3A C6 59 DB 3D D4 36 A1 F9 C9 3F FE 3F	< Á < y : Č Y Ü = Ó € ° Ÿ É ? t ?
22 FE ED 56 7F FF ED CA 00 00 00 77 00 78 7F FE 12 34	í i V I í E w x I I 4
33 00 00 00 77 00 78 7F FF 12 34 00 00 00 77 00 78 00	w x I I 4 w x
44 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
55 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
66 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
77 00 00 76 00 77 00 78 00 00 00 00 00 00 00 00 00 00 00	v w x
88 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00 1C AB	I e
99 55 55 54 00 11 AC 7D 3F F3 40 91 40 7C 3C 92 42 0F	U U T < - } ? ó @ ' @   < ' B Í
AA 3C C2 3C 61 3A A6 59 B9 3D D4 36 A1 F9 C9 3F FE 3F	< í < a : ; Y a = Ó € ° Ÿ É ? t ?
BB FE ED 56 7C EF 7D 3F 00 00 00 77 00 78 61 4E 63 B7	í i V I d } ? w x a N c -
CC 00 00 00 77 00 78 7D 00 7D 3F 00 00 00 77 00 78 00	w x } } ? w x
DD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
EE 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 A5 FF	A
110 00 00 00 76 00 77 00 78 00 00 00 00 00 00 00 00 00 00	v w x
121 00 00 02 00 00 01 10 04 10 00 00 01 BA 00 00 1C 4D	I + J + ? M

3.8. ábra

## **4. fejezet**

### **Értékelés**

**Megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek**

## 5. fejezet

# Összefoglalás

### SZUMM

A program megalkotásához nem használok automatikus kódgeneráló és formális módszereket támogató segédeszközöket, így a fejlesztési folyamatba 1000 sorból kb.10 hiba[21] maradhat a rendszerben. Ami odafigyeléssel és teszteléssel, illetve a tervezés során elvégzendő validációval és verifikációval lejjebb vihető. Az implementáció során elkövetett meghibásodásból, pl. egy számláló rosszul megírt növeléséből, ha ráfut a vezérlés akkor hiba keletkezik, mely a felhasználó szempontjából hibajelenséget okoz. A hatásláncot a meghibásodási tényező csökkentésével és hibajelenség kialakulásának megakadályozásával lehet befolyásolni. Előbbit ellenőrzéssel és teszteléssel, utóbbit helyes kivételkezeléssel. Több hibatípus léphet fel, egyik az előre ismert, másik az előre nem ismert. Az előre ismert hibatípusokat optimálisan lehet kezelní a tervezés során, az előre nem ismertek ellen megfelelő rendszerstukatúra kialakítása szükséges. Esetünkben előre ismert hiba lehet

- egy vagy kettő csatorna kiesése
- csomagvesztés
- küldött érték beragadása
- küldött érték túl gyors változása

# 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.pdf](http://personal.us.es/imaza/papers/journals/maza_jint10_multimodal/maza_jint10_multimodal_v1.pdf), 2013. november 24., 15:00
- [4] <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868>, 2013. október 29, 14:00
- [5] [http://www.digi.com/pdf/wp\\_zigbee.pdf](http://www.digi.com/pdf/wp_zigbee.pdf), 2013. november 29, 14:00
- [6] <http://standards.ieee.org/getieee802/download/802.15.4-2011.pdf>, 2013. november 29, 15:00
- [7] <http://www.sensor-networks.org/?page=0823123150>, 2013. december 02, 22:00
- [8] <https://code.google.com/p/ardupilot-mega/wiki/Mission>, 2013. november 24., 15:00
- [9] <http://paparazzi.enac.fr>, 2013. november 24., 15:00
- [10] <http://www.micropilot.com/products-horizonmp.htm>, 2013. november 24., 15:00
- [11] [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
- [12] <http://wiki.openpilot.org/display/Doc/OpenPilot+Documentation>, 2013. november 24., 15:00
- [13] <http://qgroundcontrol.org/>, 2013. november 25., 15:00
- [14] <https://code.google.com/p/ardupilot-mega/wiki/HappyKillmore>, 2013. március 19., 16:00
- [15] <http://msdn.microsoft.com/en-us/library/system.io.ports.serialport.aspx>, 2013. március 20, 10:00
- [16] <http://msdn.microsoft.com/en-us/library/windows/desktop/ms533798.aspx>, 2013. december 3, 12:00
- [17] <http://msdn.microsoft.com/en-us/library/hh425099.aspx>, 2013. december 5, 12:00

- [18] <http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>, 2013. december 5, 12:00
- [19] <https://www.aut.bme.hu/Course/VIAUA218>, 2013. december 5, 12:00
- [20] <http://com0com.sourceforge.net/>, 2013. május 4, 10:00
- [21] <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. Függelék1