



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

Tartalomjegyzék

Kivonat	4
Abstract	5
1. Bevezetés	6
1.1. Motiváció	7
1.2. Előzmények	8
1.2.1. Az MTA SZTAKI ORCA 2 felépítése	8
1.3. Földi irányító állomás	13
1.3.1. Mi a földi irányító állomás feladata?	13
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	18
1.4.4. OpenPilot	19
1.4.5. Egyéb megoldások	20
1.4.6. Összehasonlítás	21
2. Kialakítás	23
2.1. Kommunikáció megvalósítása	23
2.2. Adatok fogadása	23
2.2.1. Fogadás protokollja	23
2.2.2. Fogadott adatok hibadetektálása	25
2.3. Adatok küldése	25
2.3.1. Küldés protokollja	26
2.4. Grafikus felület	27
2.4.1. Főképernyő	27
2.4.2. Tervezés képernyő	28
2.4.3. Diagnosztikai képernyő	28
2.4.4. Terminál képernyő	29
2.5. Használt technológiák bemutatása	29
2.5.1. .NET	30

2.5.2. GMap.NET	31
3. Megvalósítás	32
3.1. Program funkcióinak megvalósítása	32
3.1.1. Kapcsolódás megvalósítása	32
3.1.2. Redundáns adatok feldolgozása	32
3.1.3. Redundáns adatok hibadetektációja	33
3.1.4. Adatküldés megvalósítása	33
3.1.5. Felhasznált osztályok	35
3.2. Megjelenítési felületek	35
3.2.1. Főképernyő	36
3.2.2. Tervezés képernyő	38
3.2.3. Diagnosztikai képernyő	39
3.2.4. Terminál képernyő	40
3.3. Összefoglalás	40
4. Ellenőrzés	41
5. Összefoglalás	43
Függelék	46
F.1. Program elindítása	46
F.2. Térkép letöltése	46

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 19.

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, mezőgazdasági, meteorológiai és 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 it's low running costs and practical usage, this technology has proved that it is useful in other areas, eg. geological research, assistance to fire fighter units, agricultural, meteorological and recreational use.

MTA SZTAKI Rendszer és Irányításelméleti Kutatólaboratórium has developed control algorithms, which were elaborated into practice by an unmanned vehicle. In addition to the safe operation, it is defended with redundant hardware elements to avoid occurance of potentially active faults.

In the context of this thesys developed a software for the ground control station of this airplane, which processes and displays the redundantly sent radio signals. The ground staff can see a map-based visualization of the active waypoints and the route of the plane, while on an other screen they can see diagnostic data and faults. It is possible to create and upload a flight plan, which contains the turning points 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, 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 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 katonai és polgári repülőgépeken is egyre szélesebb felhasználási teret nyert. Ennek oka, hogy fontos és célszerű is az emberi terhelés csökkentése, ezzel növelte a repülésbiztonságot és csökkentve a költségeket. Utasszállító gépek esetében például a robotpilóta elvégez minden olyan korrekciót, amelyhez azelőtt a pilóta folyamatos figyelme, koncentrációja volt szükséges. Ám hiába a fejlett eszközkháttér, mind a pilótával, mind a pilóta nélküli repülő légijárművek teljes körű automatikus üzemeltetése egyelőre távoli cél. Ma még az emberi beavatkozásnak rendelkezésre kell állnia olyan helyzetekben, amelyekre nincs előre felkészítve az az automatikus irányítórendszer. UAV-k esetében ilyen beavatkozásokhoz létfontosságú, hogy az operátor lássa a gép aktuális pozíóját, diagnosztikai adatait. Ezt a feladatot látja el az ún. földi irányító állomás, azaz GCS (Ground Control Station). Szakdolgozatom célja egy meglevő pilóta nélküli légijármű földi irányító állomásának megtervezése és elkészítése, a megbíz-

ható működés követelményeinek figyelembe vételével.

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 légiármű stabil repülési tulajdonságait garantáló algoritmusok kidolgozásához és implementálásához elengedhetetlen a kormányszervek harmonikus mozgatása és a tolóerő szabályozása. Abban az esetben, ha a repülőgépet feladatokkal látjuk el, pl. fordulópontokat követve feltérképezni egy előre kijelölt 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 légiármű projekt” keretében egy olyan avionikai rendszer fejlesztése is folyik, amelynek célja minden egyes repülőgép alrendszer meghibásodásának diagnosztizálása és ezen diagnosztikai információkat felhasználva a repülőgép átkonfigurálása.

Feladatom tehát 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. Feladatom tehát 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. A repülőgép 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, 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 jelenlegi repülő egy az MTA SZTAKI munkatársai által épített 3.2 m fesztávolságú modell, amelybe diagnosztikai és hibadetekciós célokra egyedi tervezésű komponenseket szereltek be. Figyelembe véve, hogy a kereskedelmi forgalomban beszerezhető szervő motorok nem szolgálnak elég információval a kitérésükéről, 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 légi jármű” projektjében elsődleges szempont, hogy egy komponens meghibásodása ne okozza a gép vesztét, tehát a rendszerben ne maradjon SPOF (Single Point Of Failure)¹. Ezt redundanciával érhetjük el, azaz a repülőgépen duplikáljuk azokat az elemeket, melyek nélkülözhetetlenek a levegőben maradáshoz:

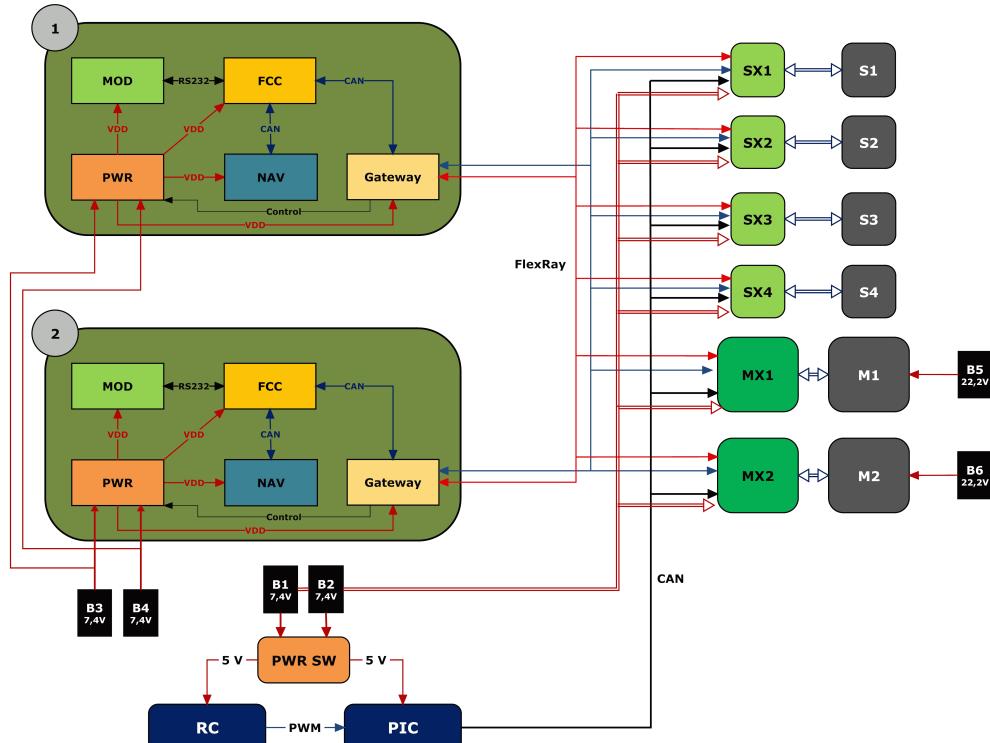
- motor
- kormányfelületek és ezeket mozgató motorok
- tápellátás
- központi számítógép

A 1.2. ábrán látható a kialakított architektúra. Elemei:

- **FCC** (Flight Control Computer) repülőgép irányítása
- **NAV** (Navigation) GPS, nyomásmérő, orientációmérő

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

- **MOD** (Modem) modem, kommunikáció
- **Gateway** CAN-FlexRay átjáró
- **PWR** (Power) feszültségválasztó
- **S1, S2, S3, S4** (Servo) kormányfelületeket mozgató szervó motorok
- **SX1, SX2, SX3, SX4** szervó motorok szabályzóelektronikája
- **M1, M2** (Motor) motorok
- **MX1, MX2** motorok szabályzóelektronikája
- **RC** (Remote Control) távvezérlő egység, manuális vezérlés
- **PIC** (PIC Microcontroller) PWM (Pulse-Width Modulation)²-CAN (Controller Area Network)³ 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-ból érkező magasság és pozíció adatokat, illetve egy, az ábrán nem

²impulzusszélesség moduláció, mely a szabályzást nem feszültségszint növelésével, hanem annak időtartamával éri el

³egy számítógépes hálózati protokoll és adatbusz szabvány melyet mikrokontrollerek és egyéb gazdaszámítógép nélkül működő eszközök kommunikációjára terveztek [2]

látható IMU (Inertial Measurement Unit)⁴-ból érkező orientáció és gyorsulás értékeket. A navigációs egységen 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 kormányfelületeket mozgató szervó motorok és a meghajtásért felelős motorok. Ezek az elemek 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 megfelelően működhesse. Ezekre bármilyen szabályzóalgoritmus írható, hibadiagnosztikai célokra fault detection filter-t vagy Kármán szűrő alkalmazható. A szervók mágneses enkódere a kitérésről, a motorok elektronikája a fogyasztásról ad információt.

Hibatűrésből 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 jelzésű akkumulátorok a legnagyobb fogyasztású egységeket, azaz a motorokat hivatottak kiszolgálni, ezért ezek kapacitása a legnagyobb. A B1 és B2 akkumulátor biztosítja a beavatkozó szervek, és a hozzájuk tartozó vezérlőelektronikáknak az áramforrást.

A repülőgép jelenleg csak távirányítással tud felszállni, így a távvezérlő egység és a PWM-CAN átalakításért felelős PIC egység is ezt a két akkumulátort használja. A PIC közvetlenül a szabályzó elektronikákhoz 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 modemelek [3] (lásd 1.3. ábra), melyek alacsony fogyasztásuk és nagy hatótávolságuk miatt ideálisak egy szűkös energiaforrással rendelkező robotrepülőbe. Ugyanilyen modemelek csatlakoznak a földi irányító egységhez, melyek a fedélzeti modemelekhez hasonlóan soros porton keresztül küldik a vett jeleket. A modempárok közötti vezeték nélküli protokoll: 802.15.4, amivel a 1.2.1 fejezet foglalkozik.

A kiválasztott XBee-PRO 868 modem kétféleképpen képes kommunikálni a csatlakoztatott eszközzel:

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

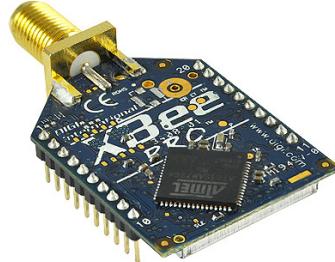
AT: Alapértelmezetten transzparens módon zajlik a kommunikáció a modemhez csatlakoztatott eszköz és a modem között [4]. Ennek lényege, hogy a csatlakoztatott eszköz

⁴orientáció- és gyorsulásmérő berendezés

⁵előre megírt komponensek használatához biztosít interfész

számára a kommunikáció ugyanúgy zajlik, mintha soros protokollon keresztül megvalósított közvetlen összeköttetésben állna másik eszközzel. A modem a soros portján bejövő üzeneteket rádiójelekké alakítja, melyet a virtuális kapcsolat végpontja fogad és visszaalakít. Egy pont-pont kapcsolat kiépítése ebben a módban a legegyszerűbb.

API: API mód lényege, hogy a csatlakoztatott eszköz a modem által biztosított API-n keresztül kommunikál. Az AT módhoz képest nagyobb rugalmasságot biztosít, mivel egy csomag a programozó szándéka szerint tartalmazhatja a küldő és fogadó eszköz címét, a csomag típusát, checksum-ját. A küldő fél a csomagok megérkezéséről ACK (Acknowledgement)⁶ jelzéssel bizonyosodhat meg, ennek elmaradásnál újból küldés lehetséges. A csomagok többletinformációjával lehetséges broadcast üzenetek⁷ küldése, egy-több kapcsolat 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 alcsoporthoz 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)⁸ modell első (fizikai) és második (adatkapcsolati) rétegét valósítja meg (1.4. ábra), erre építkezik a ZigBee [6] cégg 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 összeköttest teremt a hardverrel, specifikálja a működési feszültséget, szabályozza a használandó frekvenciát, ami Európában 868-868.6 MHz, Észak-Amerikában 902-928 MHz, világszerte 2.4-2.4835 GHz-es ISM (Industrial, Scientific and Medical)⁹ tartományban helyezkedik el.

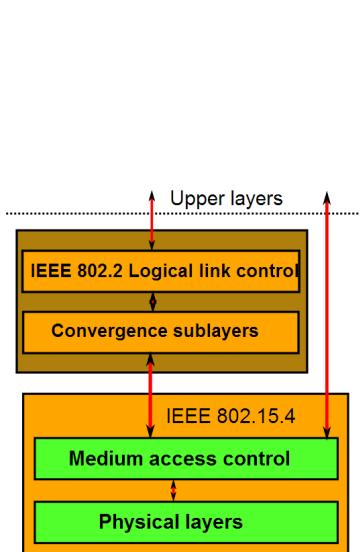
Az adatkapcsolati rétegnek a feladata hibamentes átvitel biztosítása két pont között hibajavítással, hiba jelzéssel és forgalomszabályozással. A biteket keretekbe ágyazva küldi a felsőbb rétegnek. Egy keret maximális mérete 127 bájt, formátuma a IEEE 802.15.4-2011-es szabványban specifikált.

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

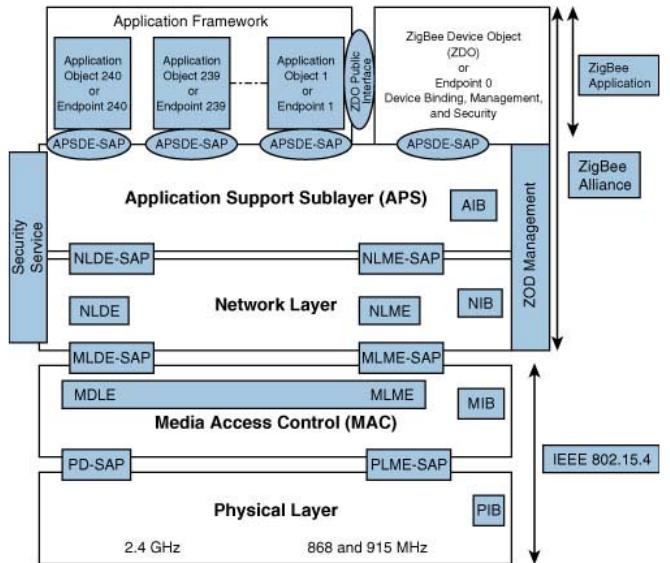
⁷olyan üzenet, melyet egy csomópont az összes vele kapcsolatban álló csomópontnak elküld

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

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



1.4. ábra. IEEE 802.15.4 protokoll rétegei



1.5. ábra. ZigBee protokoll

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

Az alkalmazási réteg összeköti a hálózati réteg adatkommunikációs és menedzsment részét a ZDO (ZigBee Device Objects)-k között. A ZDO-k a kapcsolódó eszközök azonosításában és a kommunikáció titkosításában segédkeznek.

A protokoll jellemzői

Alacsony fogyasztás: A Zigbee protokollra épülő eszközök alacsony fogyasztása abban rejlik, hogy alvó állapotból 30 ms alatt aktívrá 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] 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 úgynevezett 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 egy program formájában egy olyan grafikus felhasználói felülettel ellátott földi állomás elkészítése, ami képes a redundáns kommunikációs csatornák adatainak kezelésére és hatékony megjelenítésére. 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. Ilyen lehet például egy 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 találnom. Az esetleges eltérések kiértékelését és az ezen alapuló hibadetektációt meg kell oldanom.

- Biztonságos protokoll:

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

1.3.3. Grafikus felhasználói felület

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

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

1.4. Földi irányító állomások bemutatása és összehasonlítása

Számos megoldás született földi állomások GUI (Graphical User Interface)¹⁰-jainak kialakítására. Elsődleges követelmény, hogy az operátor minden a legfontosabb információkat láthassa, ehhez szoftverergonómiaiag kell 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 bemutatom a piacon lévő megjelenítési felületeket és megoldásokat, melyeket a fejezet végén összehasonlítok.

1.4.1. ArduPilot

Az Ardupilot projekt [11] létrejöttének oka, hogy otthoni körülmények között, nem ipari alkatrészektől 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.

¹⁰grafikus megjelenítés

Főképernyő

A program funkcionálása 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.



1.6. ábra. *MissionPlanner*

Ez a nézet akkor jöhet 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										
1	TAKEOFF		▼	0	0	0	0	10,8333060	-14,0625000	100	X	
2	WAYPOINT		▼	0	0	0	0	4,5654736	63,2812500	100	X	
3	RETURN_TO_LAUNCH		▼	0	0	0	0	4,5654736	63,2812500	100	X	
4	LAND		▼	0	0	0	0	-25,4829512	3,1640625	100	X	

1.7. ábra. MissionPlanner tervezőképernyő, pontok típusai

A program jobb felső sarkában a „Csatlakozás” gomb minden 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

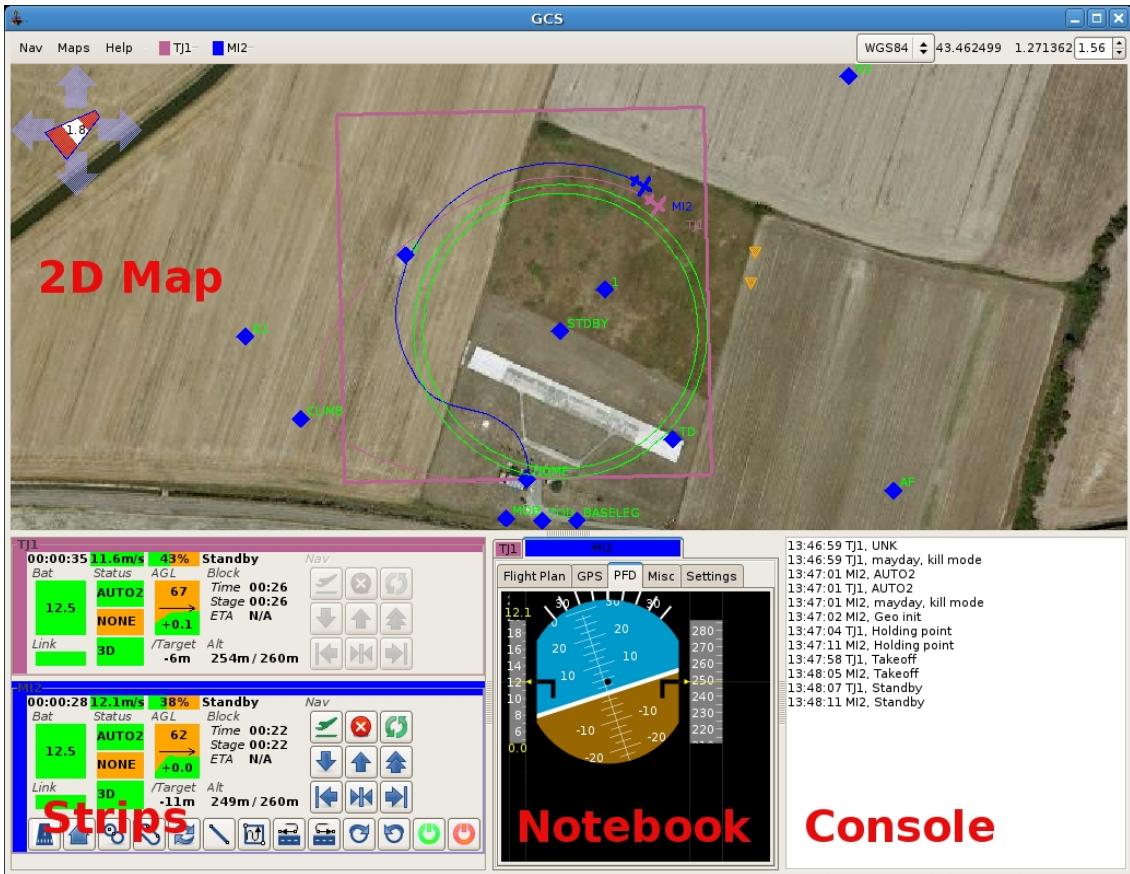
Az ArduPilot-hoz hasonlóan a Paparazzi projekt [13] is a könnyen, otthon összeszerelhető repülő megépítése jegyében született. Nem csak egy nyílt forráskódú robotpilótát ad, hanem egy komplett „csináld magad” készletet, amelyben a feszültségszabályzótól kezdve a GPS vevőig minden biztosított, a hozzá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

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 útvonala (különböző színnel jelölve) és a fordulópontok láthatóak. A



1.8. ábra. *Paparazzi program*

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.

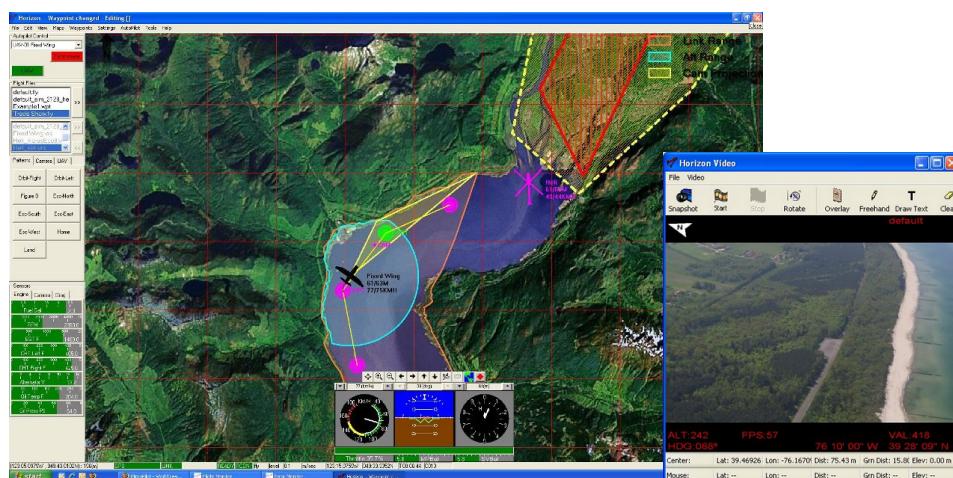
Összegzés

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

1.4.3. MicroPilot Horizon

„Az 1995 óta működő kanadai MicroPilot [14] a világ egyik legismertebb robotpilóta gyártó cége, 65 országban több mint 750 alkalmazó használja az eszközeiket. Népszerűek az iskolákban, egyetemeken, kutató intézetekben, a gazdaság különböző területein, de a védelmi szféra is szép számmal használ robot légi járművein MicroPilot berendezéseket. Az adatátviteli csatorna ugyanazokat a funkciókat képes biztosítani, mint a programbevitelnél használt kábeles összeköttetés, így ezen keresztül repülés közben is módosítható az útvonal, a repülési paraméterek és az egyéb beállítások. A MicroPilot fedélzeti egysége ezen kívül egy rádió távirányító vevőberendezést is fogad, amely lehetőséget teremt a földi adó konzoljáról az irányítás átvételére és botkormányos kézi vezérlésre. Ezt alapvetően a fel és leszállás idejére, illetve az útvonal kritikus szakaszain használják. Amennyiben az RC távirányítóval működő repülőgép veszti el a kapcsolatot, akkor a fedélzeti vevőberendezés Failsafe üzemmódra kapcsol és annak beállítása szerint működteti a repülőgépet. A Failsafe vagy az utolsó szervó állást őrzi meg, vagy egy előre programozott legbiztonságosabb földet érést ígérő beállításra ugrik.”[15]

A hozzá készített földi irányító egység a MicroPilot Horizon nevet viseli.



1.9. ábra. 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 eszközökhez 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)¹¹ gomb, mellyel egy pozíció későbbi kivizsgálásra megjelölhető, ha esetleg valami olyat lát az operátor, melyet érdemes felkeresni.

1.4.4. OpenPilot

Az OpenPilot projekt [16] célja, hogy nyílt forráskódú, magas minőségű robot pilótát kínáljon autonóm légijárművek 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 elmenthetők és visszatölthetők, így igazodva különböző a konfigurációk igényeihez.

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.

¹¹GPS koordinátával megjelölhető érdekes hely



1.11. ábra. *OpenPilot főképernöke*

1.4.5. Egyéb megoldások

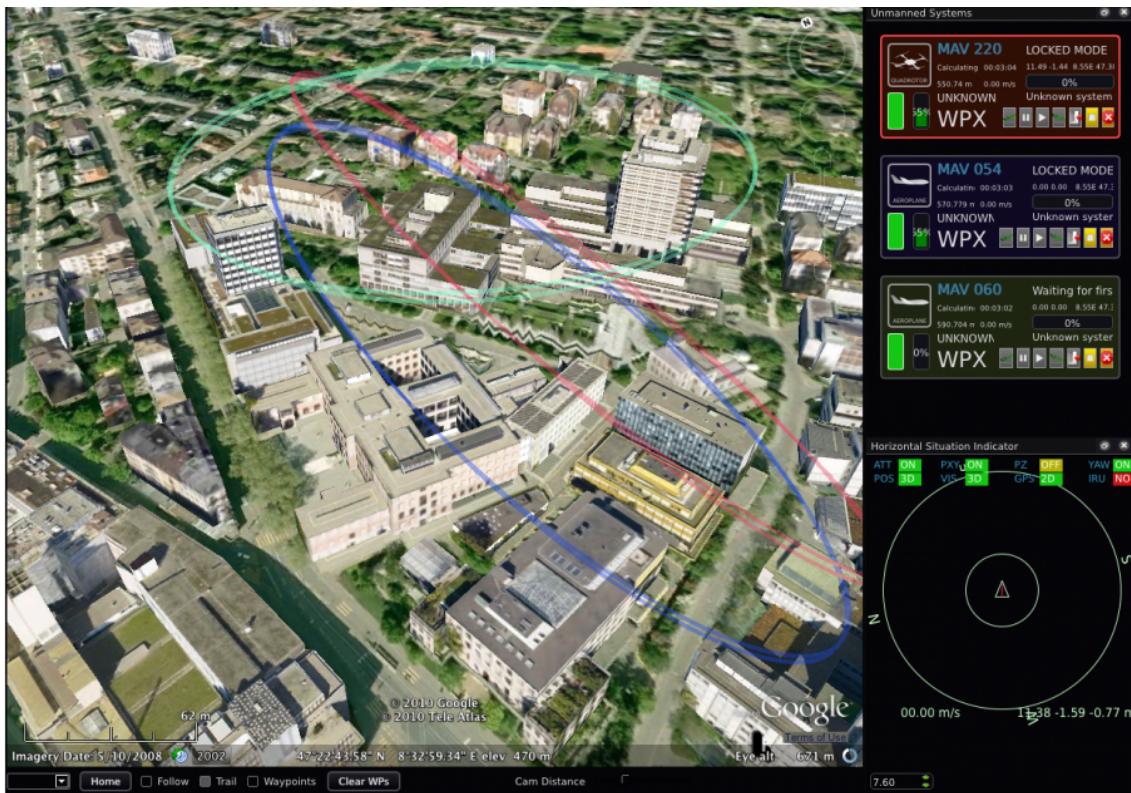
QGroundControl

Ez a program [17] nyílt forráskódú és a MAVLink¹² protokollt használja a csatlakoztatott eszközzel történő kommunikációra. Érdekesség, hogy a program a repülőgép aktuális pozícióját a 1.12. ábrán 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.

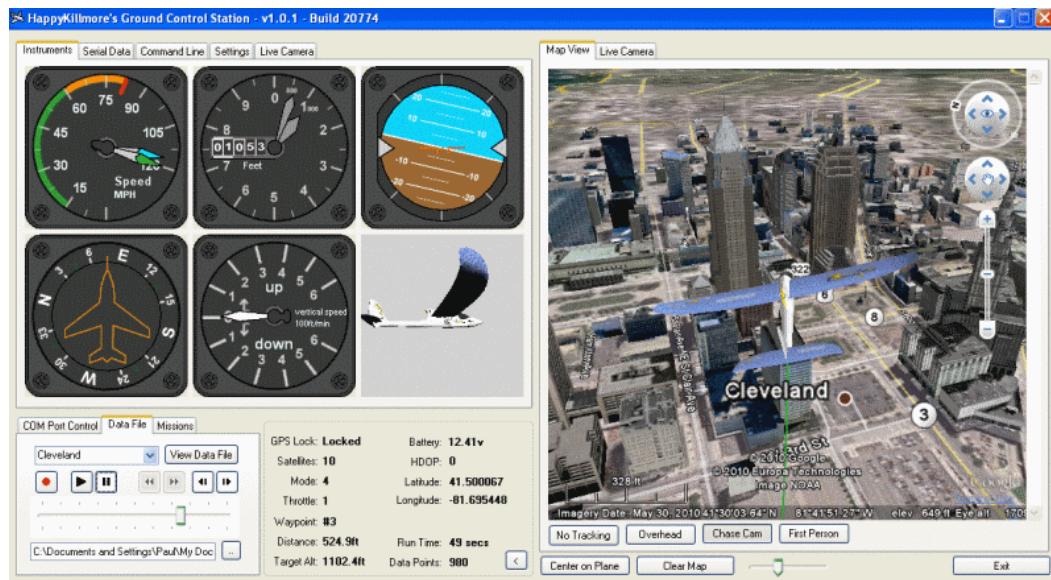
HappyKillmore

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 (1.13. ábra). 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.

¹²pehelysúlyú protokoll, C nyelvű header fájlok írják le. Változó csomagformátumú, egy csomag tartalmazza a tartalmának típusát, melyet a fogadó oldal felhasznál az adattartalom értelmezéséhez.



1.12. ábra. *QGroundControl* program



1.13. ábra. *HappyKillMore* program

1.4.6. Összehasonlítás

A fentebb bemutatott földi irányító állomások tulajdonságait a 1.1. táblázatban foglaltam össze. Látható, hogy megoldások forráskódja elég változatos a választott nyelv tekintetében. A nyílt forráskódúak az összes napjainkban használat programozási nyelvet felhasználják. Ezek tanulmányozásával ötleteket lehet meríteni a tervezéshez és megvalósításhoz.

Az összes vizsgált megoldásra jellemző, hogy a repülőgép pozíciója valamelyen térképen kerül megjelenítésre. Általában a főképernyőn a sebesség és a magasság adatok minden láthatók, így az elkészítendő felületre célszerű egy térképes megjelenítést és a legfontosabb információk kijelzését megoldani.

Számos program különbözőként használta az útvonal-kijelölés felületét a főképernyőtől, így ebből merítve készíthető egy olyan felület, melyen útvonalpontok hozzáadhatóak, módosíthatóak és feltölthetők.

Hasznos funkció az útvonal repülés közbeni módosítása, ezért érdemes ezt a lehetőséget beépíteni az elkészítendő programba.

Néhány megoldás több eszköz párhuzamos kiszolgálását támogatta, jelenleg a feladatom egy eszközhöz való csatlakozás megoldása, ám ha szükséges, látható, ez nem egy megoldhatatlan probléma.

Egyik sem alkalmaz redundanciát a kommunikáció megvalósítására, így a dolgozat szempontjából érdekes hibadetektációra nem találtam példát. A repülőgép, melyhez a programot készíttem, nagy megbízhatóságú, így adatai duplikált csatornán küldi. Ezért a két port kezelése és az azokon érkező adatok feldolgozása megoldandó probléma.

A MicroPilot program (1.4.3 fejezet) elsődleges funkciója a repülőgép távvezérléséhez kapcsolatos információk kijelzése, ezért az operátort különféle üzenetekkel figyelmezteti például előre beállított tiltott légtér megközelítése. Ilyen figyelemfelkeltő üzeneteket érdemes használni hibadetektálás eredményének kijelzésére.

Program	Ardupilot MissionPlanner	Paparazzi	MicroPilot	QGround Control	Happy Killmore
Forráskód	C#, nyílt	?, nem nyílt	?, nem nyílt	C++, nyílt	VB, nyílt
Térképes vizualizáció	+	+	+	+	+
Tervezés képernyő	+	-	+	+	+
Több eszköz egyidejű kezelése	-	+	+	+	+
Útvonal szerkesztése „on-the-fly”	-	+	+	-	?
Figyelmeztetés	-	-	+	+	-
Redundáns hibatűrés	-	-	-	-	-

1.1. táblázat. Földi irányító állomások összehasonlítása

2. fejezet

Kialakítás

Az előző fejezetben ismertettem a feladatomat, azaz egy redundáns kommunikációs csatornák adatainak kezelésére és hatékony megjelenítésére képes grafikus felhasználói felülettel ellátott földi irányító állomás kifejlesztésének egyes lépéseiit, illetve bemutattam már létező megoldásokat, amik tanulságait felhasználók saját programom tervezése és implementációja során.

Ebben a fejezetben sorra veszem és leírom azokat a problémákat, amiket meg kell oldanom 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 kezelní 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. Fogadás protokollja

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 adatok, 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.

Telemetria csomag leírás

érték = nyers érték / skálázás - ofszet							
bájt index	leírás	tipus	bájt sorrend	változó név	offset	skálázás	mértekegység
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 2/4 3/4 4/4	get_time()	0,00	10000,00	s
7	magasság parancs	uint16	1/2 2/2	alt_dmd	0,00	0x7FFF / 10000	m
9	sebesség parancs	uint16	1/2 2/2	ias_dmd	0,00	0x7FFF / 80	m/s
11	szögsebességek	uint16	1/2 2/2	smart_imu->gyr1[0]	250,00	0x7FFF / 500,0	°/s
13			1/2 2/2	smart_imu->gyr1[1]	250,00	0x7FFF / 500,0	°/s
15	nyomás alapú magasság	uint16	1/2 2/2	smart_imu->gyr1[2]	250,00	0x7FFF / 500,0	°/s
17			1/2 2/2	tmp_P	200,00	0x7FFF / 8200	m
19	IAS	uint16	1/2 2/2	smart_imu->ias	0,00	0x7FFF / 80	m/s
21			1/2 2/2	ahrs->psi	180,00	0x7FFF / 360,0	°
23	Euler-szögek	uint16	1/2 2/2	ahrs->theta	90,00	0x7FFF / 360,0	°
25			1/2 2/2	ahrs->phi	180,00	0x7FFF / 360,0	°
27	normalizált kormánykitérítések	uint16	1/2 2/2	control_cm->dr		0x7FFF	[-1..1]-re normálva
29			1/2 2/2	control_cm->de		0x7FFF	[-1..1]-re normálva
31	normalizált gázkarállás	uint16	1/2 2/2	control_cm->da		0x7FFF	[-1..1]-re normálva
33			1/2 2/2	control_cm->dthr		0x7FFF	[0..1]-re normálva
35	GPS egészség	uint16	1/2 2/2	TRUE/FALSE			1 vagy 0
37			1/2 2/2 3/4 4/4	smart_gps->POSLH.lon	90,00	0xFFFFFFFF / 180,0	°
41	GPS pozíció	uint32	1/4 2/4 3/4 4/4	smart_gps->POSLH.lat	180,00	0xFFFFFFFF / 360,0	°
45			1/4 2/4 3/4 4/4	smart_gps->POSLH.height	100,00	0xFFFFFFFF / 10100,0	°
49	gyorsulás	uint16	1/2 2/2	smart_imu->acc1[0]	2,50	0x7FFF / 5,0	g
51			1/2 2/2	smart_imu->acc1[1]	2,50	0x7FFF / 5,0	g
53	mágneses térrösségek	uint16	1/2 2/2	smart_imu->acc1[2]	2,50	0x7FFF / 5,0	g
55			1/2 2/2	smart_imu->mag[0]	2,00	0x7FFF / 4	
57	következő útvonalpont	uint16	1/2 2/2	smart_imu->mag[1]	2,00	0x7FFF / 4	Föld mágneses tere a laborban
59			1/2 2/2	smart_imu->mag[2]	2,00	0x7FFF / 4	
61	repülési mód (manuális/auto)	uint16	1/2 2/2	flightmode			
63	következő útvonalpont	uint16	1/2 2/2	nextwaypoint*10+lc			
65	tartalék hely						
66							
67							
68							
69							
70							
71	EKF státusz	uint16	1/2 2/2	state->ms			
73	ellenőrzösszeg	uint16	1/2 2/2	checksum			
74							

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

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

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

2.2.2. Fogadott adatok hibadetektálása

Az adatok fogadása során különböző hibatípusok fordulhatnak elő:

- 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.

2.3. 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.

¹Bináris számok XOR képzésével kapott 1-esek száma

2.3.1. Küldés protokollja

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 fejlécvel (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, 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 fedélzetű programot.

A feltöltés során mindenki csatlakoztatott modem segítségével redundánsan küldöm el az előállított csomagot. Ha a csomag sérült 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(fix 'G')		
1	start	bájt(fix 'P')		
2	start	bájt(fix '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

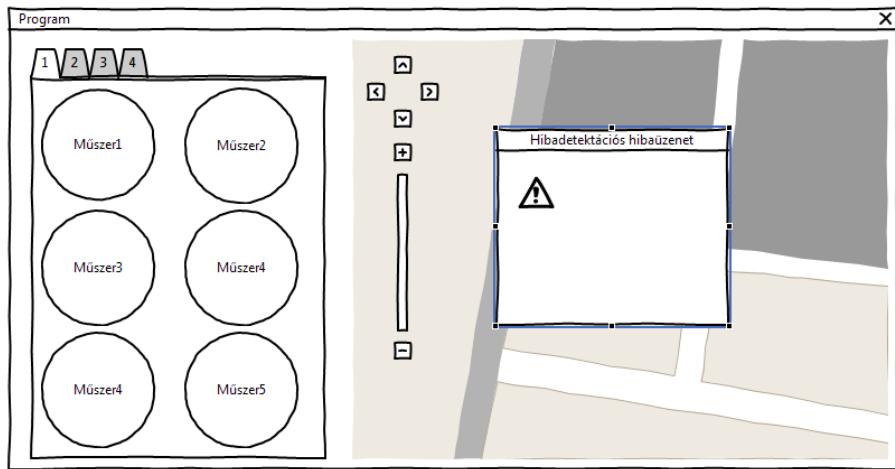
Felmerülhet a kérdés, hogy a feltöltés ezzel a protokollal elég hibatűrő-e. Mivel a fogadás protokollja is hasonló hibadetektálást biztosít, így ez a megoldás elégsgesnek tűnik. Továbbá, ha a küldés sikerességének visszajelzése elmaradna, akkor lehetőség van az üzenet újbóli elküldésére.

2.4. Grafikus felület

A 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. A különböző funkcionálitásokhoz tartozó nézetek között érdemes lehet fülek segítségével váltani. Az alábbiakban ezekre adok egy elképzelt kialakítási formát.

2.4.1. Főképernyő

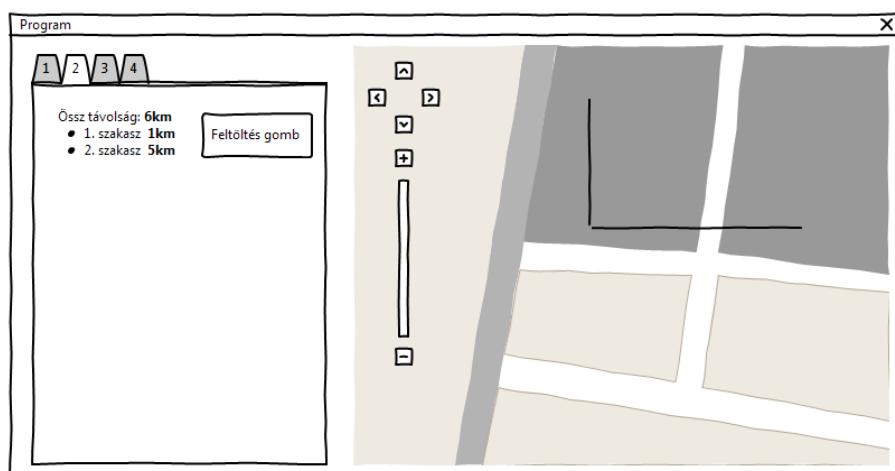
A GroundControl főképernyőjén látható a repülőgép aktuális pozíciója és iránya. A térképen a repülőgép pozíóját egy repülőgép ikon szemlélteti. Ez a rész a képernyő kb. 2/3-át foglalja el. Az oldalsó sávban a „Glass Cockpit” kerül kialakításra, ez a nézet tartalmazza a gép aktuális sebességét, iránytű 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. Az elképzelt képernyő terv a 2.1. ábrán látható



2.1. ábra. Főképernyő terve

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 megsokszorozott Google Maps-hez hasonló módon hosszan kattintva átrakhata, illetve köztes pontokat törölheti. A már létező megoldások összehasonlításából kiderült, hogy érdemes a kijelölt útvonal hosszáról tájékoztatni a felhasználót, ez látszik a 2.2. ábrán.



2.2. ábra. Tervezés képernyő terve

2.4.3. Diagnosztikai képernyő

A diagnosztikai képernyőn a két porton érkező dekódolt értékek látszódnak (lásd 2.3. ábrá) két oszlopban, mellettük egy hibaérték, mely a különböző hibatípusok hibaszámának összege.

Jel neve	Port1	Port2	Hibaszám
GPS.lat	32.1	32.1	0
GPS.lon	20.1	20.1	0

2.3. ábra. Diagnosztikai képernyő terve

2.4.4. Terminál képernyő

A 2.4. ábrán látható terminál képernyőn lehetőség nyílik a fogadott csomagok hexadecimális vagy decimális formában történő megjelenítésére, így az operátor alacsony szinten (a fogadott csomagok feldolgozatlan 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. Az elképzelt képernyő terv

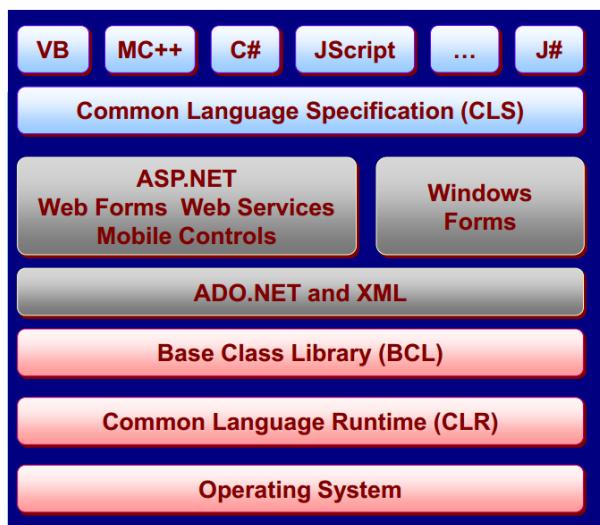
Port1-ből kapott bajtok
Port1-ből kapott bajtok

2.4. ábra. Terminál képernyő terve

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



2.5. ábra. .NET keresztrendszer felépítése

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)², 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)³ 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.

Common Language Specification

A .NET nyelvfüggetlen, így a keretrendszer szolgáltatásai hozzáérhetőek minden nyelv számára, mely nyelvi specifikációnak megfelel. Többek között ezek a nyelvek támogatottak: C#, C++, Visual Basic [22].

²közös nyelvi futtatókörnyezet

³mely előre megírt komponensek használatához biztosít interfész

Windows Forms

Grafikus megjelenítést biztosít az alkalmazásoknak, különböző elemek (beviteli mező, gomb, kép, választó lista) helyezhetők el rá. GDI (Graphic Device Interface)⁴ [23] segítségével történik a krajzolás. 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] könyvtárcsomagot 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.

⁴ Microsoft API, mely az operációs rendszer része, feladata grafikus elemek megjelenítése

3. fejezet

Megvalósítás

Az előző fejezetekben összegyűjtöttem a megvalósításhoz szükséges információkat és lépéseket. Ebben a fejezetben a konkrét implementációt mutatom be.

3.1. Program funkcióinak megvalósítása

A program megfelelő működéséhez különböző funkciók megoldása szükséges, ezek megoldását az alábbiakban írom le.

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 *DataReceived()* függvény, mely akkor hívódik meg, ha a felépített kapcsolaton keresztül adat érkezett. Jelen esetben a kapcsolat sebességén műlhet, hogy egy ilyen adatcsomagban egy egész számunkra megfelelő csomag érkezett-e. Előfordulhat az az eset, hogy a repülőgép már küldi az adatait és a program ennek az adatfolyamnak a közepébe kapcsolódik bele, így egy előző csomag eleje és a következő csomag vége lemaradhat. Ezért szükséges egy puffer alkalmazni, melynek végére minden egyes beérkező 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 érkezőeknek. 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 mindenkettő 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 ϵ_1 -nél (ez esetben 0.000001) kisebb, akkor növelem a jel hibaszámlálóját. Ha ez a számláló elér egy küszöböt, akkor gyanús a beragadás esélye. Ezt a kijelzés során piros „Beragadás” üzenettel jelzem a beragadt jel sorában.

Túl nagy ugráás: Egy új elem érkezésekor a FIFO-ban lévő elemek átlagát kiszámítom és ha az újonnan érkezett érték ϵ_2 -ször (ez esetben 0.1) 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.

3.1.4. Adatküldés megvalósítása

A visszirányú kommunikáció során a Tervezés képernyőn (3.2.2 fejezet) kijelölt útvonalhoz tartozó fordulópontok listáját párhuzamos csatornákon küldi el a program. A 3.4. ábrán látható 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 minden kettő sorra,

végső soron a repülőgépre. Mivel az útvonal maximum 10 db pontot tartalmazhat, ezért új pont lerakásánál a lista méretét a következő kódrészlettel 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 sérzetlenül megérkezett, a repülőgép az aktuális kommunikációs csatornáján egy „ACK” üzenettel jelez. Ezt fogadva az operátort egy tájékoztató ablak formájában (3.1. ábra) tájékoztatja a program.

Az „ACK” jelzést fogadó programrészlet, mely kiváltja a tájékoztató ablak megjelenítését:

```
int i = 0;
for (; i < pufferB.Count; i++)
{
    if (pufferB[i] == 'A' && pufferB[i + 1] == 'C' && pufferB[i + 2] == 'K')
    {
        int k = 0;
        for (int j = i; j < i + 3; j++)
        {
            receivedBytesB[k++] = pufferB[i];
            pufferB.RemoveAt(i);
        }
        pufferB.RemoveRange(0, i);
        MessageBox.Show("Sikeres feltöltés: " + serialPort2.PortName,
                       "Feltöltés sikeres", MessageBoxButtons.OK,
                       MessageBoxIcon.Information);
        break;
    }
}
```

A porthoz tartozó pufferen iterálva megkeresem az ACK üzenetet (ha van) és törlöm az ACK üzenetet hordozó 3 bájtöt. Ez után egy felugró ablakot jelenítek meg, mely tar-

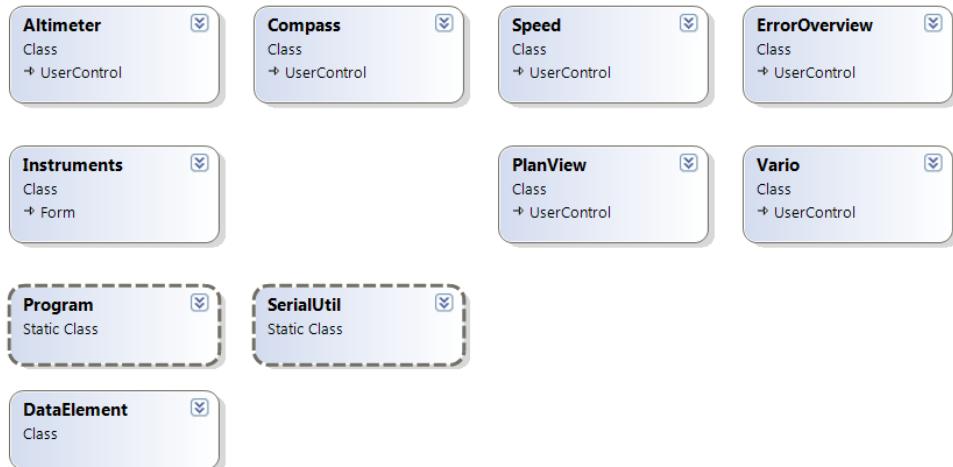
talmazza a port nevét és egy „Feltöltés sikeres” üzenetet.



3.1. ábra. Sikeres feltöltés tájékoztató ablaka

3.1.5. Felhasznált osztályok

A 3.2. 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 *UserControl*¹-ök megjelenítéséről 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 adattagját egy *DataElement* objektumba helyezi, mely egyben a hibadiagnosztikát is megoldja.



3.2. ábra. Felhasznált osztályok

3.2. Megjeleníti felületek

Felhasználói szempontból a grafikus megjelenítés a legérdekesebb, ez ha nincs megfelelően elkészítve a fentebb tárgyalt megoldások akár értelmüket is veszthetik. Ezért a felület

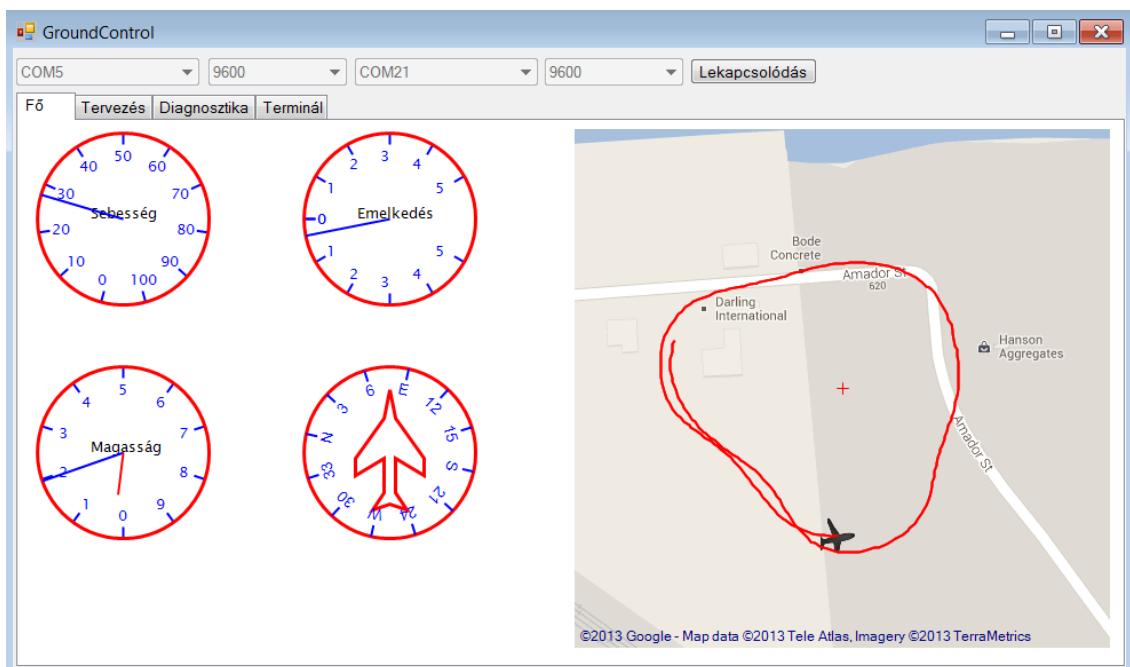
¹olyan összetett grafikus elem, ami tervezési időben vizuálisan elkészíthető, pont úgy, ahogy egy űrlap. Haszna, hogy űrlapokra, illetve más UserControlokra lehet elhelyezni.

kényelmes kezelhetőségére és az ablakok elhelyezésére ügyelni kell. A továbbiakban az el-készített 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 *UserControll*-jait melyek tervező nézetben „Drag and Drop” technológiával a megfelelő helyre húzhatóak és könnyen beköthetőek.

A 3.3. 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 látszódik, a méret módosításával ezek fix pozícióban maradnak. A térkép a program jobb alsó sarkához van horgonyozva, így átméretezéskor a térkép területe is változik.



3.3. 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 újrarendezése, 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 kijelezen-

dő 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(-textSize.Width / 2, -textSize.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 mindenkorán 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 nyomvonalat. Erré a rétegre kerül a repülőt szimbolizáló *planeMarker* ikont is, aminek pozícióját minden 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”² 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

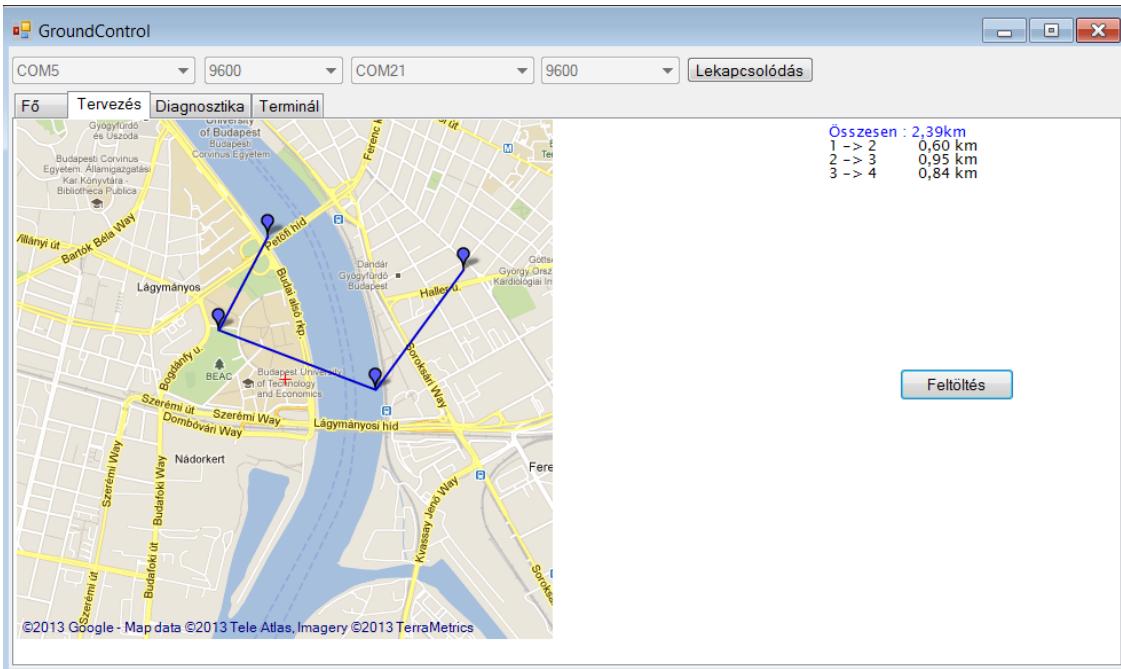
A tervező képernyőn (3.4. ábra) lehetséges az útvonal kijelölése és feltöltése. Új útvonalpont a felületen dupla egérkattintással rakható le, mely hosszan nyomva szerkeszthető, jobb kattintással törölhető. A pontokat összekötő szakaszok hossza jobb oldalt megjelenik és a tervezhetőség kedvéért összegzésre is kerül.

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 kényelmessé válik. A mozgatást végző programrészlet:

```
private void gmap_plan_MouseMove(object sender, MouseEventArgs e)
{
    if (e.Button==MouseButtons.Left && isDraggingMarker && currentMarker!= null)
    {
        currentMarker.Position = gmap_plan.FromLocalToLatLng(e.X, e.Y);
        PointLatLng newPos = gmap_plan.FromLocalToLatLng(e.X, e.Y);
        plannedRoute.Points.RemoveAt(indexOfCurrentPoint);
        plannedRoute.Points.Insert(indexOfCurrentPoint, newPos);

        gmap_plan.UpdateRouteLocalPosition(plannedRoute);
        gmap_plan.Refresh();
        planView.Invalidate();
    }
}
```

²a repülőgép függőleges tengelye körüli elfordulás mértéke a mágneses Északi-sarkhoz viszonyítva



3.4. ábra. *GroundControl* tervező felülete

3.2.3. Diagnosztikai képernyő

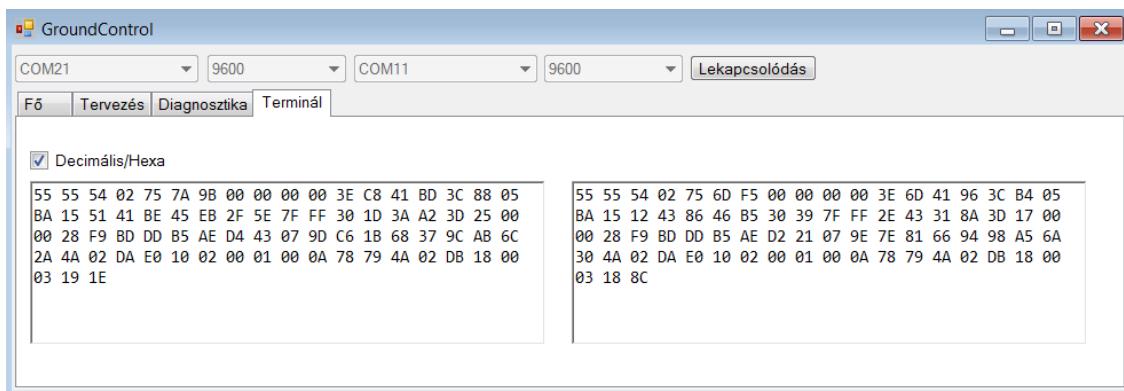
A diagnosztikai képernyőn a portokon érkező adatok és hozzájuk tatozó hibaszámláló láttható. Ha az egyik érték hibaszámlálója elér egy kritikus szintet, akkor pirossá válik és egy felugró ablakon figyelmezteti az operátort, hogy valamilyen hiba történt. Ő eldöntheti, hogy ez csak az egyik műszer meghibásodása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

Fő	Tervezés	Diagnosztika	Terminál
idő	4119,59	4119,91	000 015
magasság	0,00	0,00	1365 1365 Beragadás
sebesség	0,00	0,00	1365 1365 Beragadás
smart_imu->gyr1[0]	-121,47	-121,90	000 015
smart_imu->gyr1[1]	-125,54	-125,49	000 015
smart_imu->gyr1[2]	-125,08	-124,75	000 015
tmp_P	10,00	9,86	001 031
sebesség	6,13	6,24	000 015
ahrs->psi	-82,80	-83,66	000 016
ahrs->theta	-44,21	-44,34	000 015
ahrs->phi	-95,02	-94,69	000 015
control_cm->dr	0,50	0,50	005 1070 Beragadás
control_cm->de	0,14	0,15	000 016
control_cm->da	0,23	0,25	000 031
control_cm->dthr	0,24	0,24	000 015
GPS health	0,00	0,00	1365 1365 Beragadás
lon	-122,38	-122,38	000 015
lat	37,75	37,75	001 016
height	201,44	201,21	000 015
smart_imu->acc1[0]	-0,59	-0,66	002 017
smart_imu->acc1[1]	-0,59	-0,66	002 017
smart_imu->acc1[2]	-1,16	-1,26	004 023
smart_imu->mag[0]	-0,84	-0,84	000 015
smart_imu->mag[1]	1,42	1,42	1061 1076 Beragadás
smart_imu->mag[2]	-1,75	-1,75	000 015
flightmode	-1,75	-1,75	000 015
nextwaypoint*10+lc	1,00	1,00	1053 1068 Beragadás
state->ms	30,00	30,00	937 937 Beragadás
EKF status	3,00	3,00	1054 1069 Beragadás

3.5. ábra. *Diagnosztikai* képernyő, port kiesését mutató hibaüzenettel

3.2.4. Terminál képernyő

A fogadott csomagokat a terminál képernyő (3.6. ábra) 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. Összefoglalás

A program ellátja a kívánt funkcionálitást, sikerült megvalósítanom a repülőgép és a GroundControl közti kommunikáció kiépítését, a repülőgép pozíciójának térkép alapú vizualizációját. A fogadott jelek eltérésének hibadetektációján alapuló figyelmeztetések megfelelően működnek.

4. fejezet

Ellenőrzés

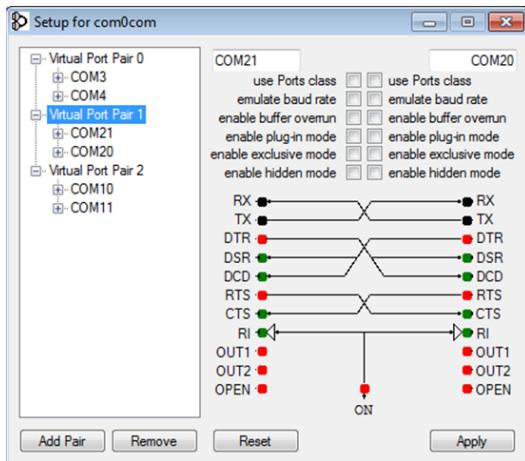
Az önálló, otthoni fejlesztést megkönnyítendő, a repülőgép HIL (Hardware In the Loop)¹ szimulációjából nyert log fájlokat használtam fel. Ennek segítségével nem volt szükséges ez idő alatt a repülő közelében lennem, a program működését elég volt csak a végleges stádiumban kipróbálni a repülőgép valódi adataival.

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 (4.1. 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 szemszögéből az adatküldés. A 4.2. ábrán egy ilyen csomag látható a 264. és 265. küldött csomag között.

¹olyan szimuláció, ami a tesztelendő beágyazott rendszernek olyan környezetet biztosít, mintha az valódi érzékelők adatait fogadná és valódi aktuátorokat irányítana



4.1. ábra. Com0com nullmodem program

```
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
973*75 bytes
```

4.2. ábra. Log fájlt feldolgozó program

A kapott log fájlok szerkezetét egy XVI32 nevű programmal tanulmányoztam. Egy csomag felépítését és méretét ezzel ellenőriztem. A 4.3. 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.

XVI32 - a.txt	
	File Edit Search Address Bookmarks Tools XVIscript Help
0	55 54 02 6F E0 DA 00 00 00 00 3F D2 42 2A 3C 3F 05 B4 16 F9 U T I o ſ Ú ? Ŋ B * < ? I ' I ū
14	14 62 41 AE 2E 91 7F FF 29 6C 3B FB 3C DE 00 00 28 F9 3E 38 I b A @ . ' I) l ; ū < T (ū > 8
28	B5 AE CE 93 07 95 23 2E 61 FB A8 7A 63 8B 4A 02 DA E0 10 02 u @ ĩ " I • # . a ū " z c < J I Ú ſ I I
3C	00 01 00 14 78 79 4A 02 DB 18 00 03 19 DA 55 55 54 02 6F EE I I I x y J I Ú I I I Ú I J U T I o i
50	22 00 00 00 40 49 42 73 3C B6 05 B0 17 C7 12 75 40 F1 2E " @ I B s < I I ° I C I u @ n .
64	A6 7F FF 27 05 3B B3 3D 62 00 00 28 F9 3C 16 B5 AE CB BB 07 I I I ' I ; I = b (ū < I u @ E > I
78	8E F9 1B 60 A3 AD 47 5A 06 4A 02 DA E0 10 02 00 01 00 14 78 Ž ū I ' E - G Z I J I Ú ſ I I I I x
8C	79 4A 02 DB 18 00 03 18 17 55 55 54 02 6F FB 6A 00 00 00 00 Y J I Ú I I I I U U T I o ū j
A0	41 50 43 4A 3D DA 05 AC 18 96 10 A6 40 A5 2E E4 7F FF 24 C3 A P C J = Ú I - I - I ; @ A . a I ' Š A
B4	3A FB 3D FE 00 00 28 F9 39 99 B5 AE C8 2C 07 87 F5 74 60 00 : ū = t (ū g @ u @ Č , I # ō t '
C8	B3 1C 4F 1A 4A 02 DA E0 10 02 00 01 00 14 78 79 4A 02 DB 18 I I I J I Ú ſ I I I I x y J I Ú I
DC	00 03 19 79 55 55 54 02 70 08 B2 00 00 00 40 6F 42 71 3D I I I y U U T I p I , @ o B q =
F0	46 05 A9 19 57 0E DD 40 FE 2F 5C 7F FF 22 B7 39 E3 3E AC 00 F I @ I W I Ÿ @ t / \ I ' " - 9 ā > -
104	00 28 F9 38 88 B5 AE C6 65 07 84 4C 98 5C 09 B5 C4 40 71 4A (ū 8 u @ Č e I I L \ u Ā @ q J
Adr. dec: 74 Char dec: 8 Overwri	

4.3. ábra. XVI nevű hexa editor

Sajnos valós körülmények között nem volt lehetőség a programot használni, mert a repülőgép jelenleg nincs abban a fejlesztési fázisban, hogy a redundáns hardver elemeket megfelelően használja. Így csak a HIL szimulációból kapott adatokra hagyatkozhattam. A repülőgép a jelenlegi állapotában csak egy modemet használ, de természetesen egy csatorna használatával is működik a program. Csatlakozás után a GroundControl érzékeli az egyik modem kiesését, mely figyelmezteti az operátort erről a hibáról.

5. fejezet

Összefoglalás

Szakdolgozatom keretében bemutattam és összehasonlítottam a piacon lévő földi irányítóegységek grafikus felületeit. Ezek összehasonlításából levont következtetéseket felhasználtam a Tervezés fejezetben.

Megterveztem, hogy a specifikáció megvalósításához milyen feladatok megoldása szükséges, ezeket a Megvalósítás fejezetben részleteztem. Bemutattam, hogy az egyes lépések megoldásához milyen eszközöket használtam fel, hogyan oldottam meg a 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 megoldtam az útvonal feltöltésének a lehetőségét is, így kényelmesen kijelölhető a repülőgép által bezárandó terület.

Továbbfejlesztési lehetőséggént a grafikus felület finomítását látom, mivel mint látható volt, már létező megoldás a beérkező jeleket grafikusan is tudta ábrázolni, mely valószínűleg hasznos lehet ebben a projektben is. Kiderült, hogy a műhorizontot, mint a repülőgép aktuális orientációját kijelző műszert az összes bemutatott program használja. Mivel ez nagyon informatív és könnyen értelmezhető megjelenítési forma, ezért ez egy hasznos funkciója lehetne a GroundControl-nak. Érdemes lenne a fedélzeten működő hibadetektációs algoritmusok eredményét is leküldeni a programnak, hiszen ezek az algoritmusok nagyobb megbízhatósággal dolgoznak, mint a vett értékek kiértékelésén alapulók.

Az MTA SZTAKI Rendszer és Irányításelméleti Kutatólaboratóriumának egy távoli célja a dolgozatban bemutatott repülőből egy olyan kooperatív flotta létrehozása, mely egymással összeköttetésben állva hajt végre feladatokat. Ennek figyelembevételével egy lehetséges fejlesztése lehetne a GroundControl több gép párhuzamos kiszolgálására való felkészítése.

Irodalomjegyzék

- [1] <http://www.azom.com/article.aspx?ArticleID=10156>, 2013. november 19., 10:00
- [2] 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, 2013. november 29., 14:00
- [7] 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, 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, 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 4 fejezetben ismertetett programok telepítését írrom le.

A program futtatásához Windows XP/Vista/7 és .NET 4.5-s verzió szükséges. A .NET keretrendszer telepítője a mellékleten *dotNetFx45.exe*¹ néven szerepel.

A repülőgéppel történő kommunikáció szimulálásához szükséges egy nullmodem telepítése. A telepítést követően létre kell hozni a COM10-COMXX és COM20-COMYY párokat. 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, az XX-szel és YY-nal jelölt portok szabadon választhatók.

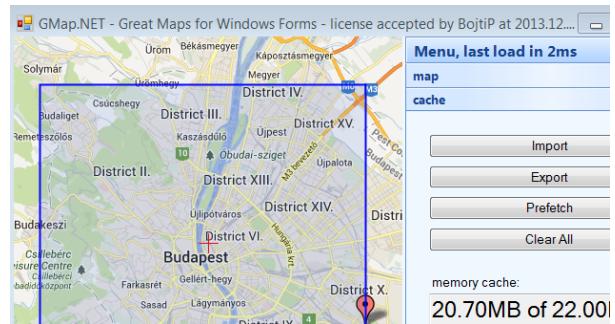
Ha kész az összeköttetés, akkor a *sorossendA.exe* és *sorossendB.exe* futtatásával a log fájlok a soros portokra íródnak. A *GroundControl.exe* program ezek után indítható, a portok kiválasztását követően látható a tényleges működése. A kapott log fájlok San Francisco repterén készültek HIL (Hardware In the Loop)² szimulációval.

F.2. Térkép letöltése

Új terület letöltésére a melléklet *GMap.Net/Demo.WindowsForms.exe* programja használandó. A program térképes felületén új területet Alt billentyű hosszan nyomásával és egér mozgatásával lehet kijelölni. Ezt követően, a „cache” fülön a „Prefetch” gomb hatására kiválasztható részletezettséggel letöltődik. Az „Export” gombra kattintva a letöltés helyéül a */map/TileDBv5/en/* útvonalon található *Data.gmdb* fájlt kiválasztva felülírható az offline térképszelvény.

¹Internet elérés szükséges

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



F.2.1. ábra