CHALMERS

4G-baserad långdistansdrönare

Utveckling av en autonom drönare för långdistansflygning

Kandidatarbete vid data- och informationsteknik

ERIK ARNEBRO SÖDERBERG
FILIP EDVINSSON
ALRIK KJELLBERG
JONATHAN CARBOL
ADAM FRITHIOFSON
OLLE NILSSON

Institutionen för data- och informationsteknik

CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG, SWEDEN 2020
www.chalmers.se

KANDIDATARBETE 2020 - DATX02-86

4G-baserad långdistansdrönare

Utveckling av en autonom drönare för långdistansflygning

ERIK ARNEBRO SÖDERBERG
JONATHAN CARBOL
FILIP EDVINSSON
ADAM FRITHIOFSON
ALRIK KJELLBERG
OLLE NILSSON

CHALMERS

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2020

4G-baserad långdistansdrönare
Utveckling av en autonom drönare för långdistansflygning

Erik Arnebro Söderberg	Jonathan Carbol
Filip Edvinsson	Adam Frithiofson
Alrik Kjellberg	Olle Nilsson

Handledare: Roger Johansson, Institutionen för data- och informationsteknik
Examinator: Arne Linde, Institutionen för data- och informationsteknik

Kandidatarbete 2020
Institutionen för data- och informationsteknik
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2020

Sammandrag

Följande rapport beskriver utvecklingen av en autonom drönare med möjlighet att starta och landa vertikalt samtidigt som den bibehåller hög effektivitet vid långdistansflygning, samt en tillhörande mobilapplikation som möjliggör övervakning av drönaren. All kommunikation med drönaren sker över 4G för i princip obegränsad räckvidd. Syftet var att drönaren skulle kunna användas för autonom transport av små tidskritiska paket över större avstånd än en radiosändare klarar av att styra.

Drönaren utvecklades från ett färdigköpt radiostyrt flygplan, på vilket fyra motorer hade monterats på vingarna för att kunna starta och landa vertikalt. För att drönaren skulle kunna ha autonoma förmågor utrustades den med en avancerad flygdator och en Raspberry Pi, som i sin tur var uppkopplad till 4G-nätet via ett 4G-modem. Detta tillät anslutning till drönaren via en VPN-tunnel för styrning samt övervakning via applikationen som utvecklades med hjälp av Android Studio.

Projektet resulterade i en autonom drönare som kan styras och övervakas över 4G, samt fungerar väl som quadcopter och flygplan. Både vertikal start och landning så väl som framåtflygning och övergång däremellan har testats och fungerar. Applikationen tillåter användaren att följa och övervaka drönaren i realtid via både karta och videoström.

Projektet lägger en god grund för fortsatt arbete kring drönares användning vid långdistanstransport av mindre föremål och förslag på vidareutveckling presenteras i rapporten.

Nyckelord: Drönare, Quadplane, VTOL, 4G, Autonom flygning, Navio2, ArduPilot, Transport, Raspberry Pi, Android

Abstract

This report describes the development of an autonomous drone with the capability to take off and land vertically while retaining high cruise efficiency, allowing for long distance flights. In addition, a companion application was developed to enable remote surveillance of the drone. All communication with the drone takes place over the 4G network for virtually unlimited reach. The scope of this project was to develop a drone that could be used for autonomous transport of small, time-sensitive packages over greater distances than one using a standard radio transmitter is capable of.

The drone was developed using an RC-plane frame on which four additional motors for vertical take-off and landing were mounted. For the drone to have autonomous capabilities, an advanced flight computer was installed onto a Raspberry Pi, which in turn was connected to a 4G modem. This setup allowed for remote connection to the drone via a VPN to monitor it through the application developed using Android Studio.

The project resulted in a functional drone with autonomous flight capabilities that can be operated and monitored over the 4G network. Vertical take-off and landing, as well as forward flight and the transition therebetween, were tested to ensure adequate performance. The application allows the user to track and monitor the drone in real time on a map, as well as through a video feed.

Furthermore, this project lays a solid foundation for further development in this area and suggestions of such possibilities are presented at the end of the report.

Keywords: Drone, Quadplane, VTOL, 4G, Autonomous flight, Navio2, ArduPilot, Transport, Raspberry Pi, Android

Förord

Vi vill tacka vår handledare Roger Johansson som bistått oss med goda råd under projektets gång och Lars Norén som hjälp oss att införskaffa nödvändig utrustning för att genomföra projektet. Vidare tackar vi Arne Linde som hjälpte oss att styra om projektet då Coronapandemin förhindrade vår planerade arbetsgång.

Ordlista

API "Application programming interface", ett gränssnitt för kommunikation mellan mjukvaror.

ArduPilot kraftfull autopilot-mjukvara med öppen källkod.

CAD "Computer-aided design".

ESC "Electronic speed control", elektronisk hastighetskontroller. Reglerar hastighet på en elektrisk motor.

GCS "Ground Control Station", markstation för styrning av drönare, vanligtvis en programvara som körs på en bärbar dator.

GStreamer fritt ramverk för hantering av multimedia.

HTTPS "Hypertext Transfer Protocol Secure", webbprotokoll med kryptering.

JSON "JavaScript Object Notation", format för överföring av strukturerad data.

Navio 2 avancerad flygdator.

Pixhawk 4 avancerad flygdator.

PX4 kraftfull autopilot-mjukvara med öppen källkod.

Quadcopter obemannad helikopter med fyra propellrar.

Quadplane Luftfarkost med vingar, framåtdrivande motorer och fyra propellrar för vertikal start och landning.

Raspberry Pi dator i bankkortsstorlek.

RTSP "Real Time Streaming Protocol", ett protokoll för strömmande av media med låg fördröjning.

SDK "Software Development Kit", en uppsättning verktyg för underlätta mjukvaruutveckling.

TCP "Transmission Control Protocol", transportprotokoll med handskakning och leveransgaranti.

UDP "User Datagram Protocol", transportprotokoll utan handskakning eller leveransgaranti.

VPN Virtuellt privat nätverk. Används för att skapa en förbindelse mellan två punkter i ett nätverk.

VTOL "Vertical take-off and landing", vertikal start och landning.

Innehåll

1 Introduktion

Privata drönare har idag blivit väldigt populära. Den formella definitionen av drönare är ett flygande fordon utan mänsklig pilot [1]. I vardagligt bruk är det mer regel än undantag att en specifik drönartyp, quadcoptern, åsyftas. Förenklat fungerar quadcoptern, som namnet antyder, på ungefär samma vis som en helikopter men med fyra propellrar som genererar lyftkraft. En nära besläktad sorts drönare är ett så kallat quadplane, som utöver de fyra propellrarna har vingar och en ytterligare propeller för att driva quadplanet framåt och diverse roder. Quadplanes är i och med vingarna bättre lämpade att färdas längre sträckor än en quadcopter.

Drönare som används i privat bruk (främst för fotografering och videotagning) styrs i regel med radiovågor vilket leder till en begränsad räckvidd. Detta innebär även att drönarens användningsområde begränsas av räckvidden. Genom att utforska 4G-uppkoppling som ett alternativ till radiostyrning kan drönarens räckvidd, och därmed användningsområde, vidgas.

Ett exempel på vad den förbättrade räckvidden skulle kunna användas till är leverans av diverse föremål. Drönare har en fördel gentemot traditionell transport med vägfordon då drönaren både tar en kortare väg (fågelvägen) och undviker trafik [2]. En begränsning att ha i åtanke är dock vikten av föremålet drönaren ska transportera. Eftersom lyftkraften för drönare är begränsad är de olämpliga för leverans av tyngre föremål.

Med tanke på en drönares begränsade lastkapacitet lämpar sig drönaren bättre till vissa användningsområden än andra. Därför betraktas ett hypotetiskt scenario i Göteborg, närmare bestämt att flyga ett blodprov mellan Östra sjukhuset och Sahlgrenska sjukhuset. Mellan dessa är det ca $7\,\mathrm{km}$ fågelvägen [3]. Detta skulle kunna leda till effektivare leverans än dagens system med kurirer mellan vårdinrättningar [4].

1.1 Syfte

Syftet med projektet är att konstruera en drönare för transport och leverans av små tidskritiska paket över större avstånd där en vanlig radiosändare inte skulle kunna användas för styrning. Drönaren ska kontrolleras via 4G med inslag av autonomi och är tänkt att användas inom miljöer där fågelvägen är överlägsen marktransport, exempelvis tät stadstrafik eller skärgårdsmiljö.

1.2 Problemformulering

Målet är att vid projektets slut kunna visa upp en drönare som klarar av att genomföra ett leveransuppdrag från Östra sjukhuset till Sahlgrenska sjukhuset helt autonomt, inklusive start, avlämning av gods och landning enligt en rutt som programmerats in på förhand. Drönaren ska under färd kunna övervakas via en videoström så en operatör kan ingripa vid oväntade händelser och med manuell styrning undvika diverse olyckshändelser. All kommunikation med drönare ska ske över 4G-nätet. Drönaren ska också automatiskt kunna vända om och återfå täckning om den skulle flyga in i ett område där uppkopplingen bryts.

1.3 Kravspecifikation

För att avgöra om projektet uppfyller de uppsatta målen bör följande kravspecifikation uppfyllas.

1.3.1 Drönare

- Drönaren kan styras och övervakas via 4G.
- Drönaren kan starta och landa vertikalt (VTOL).
- Drönaren kan flyga horisontellt.
- Drönaren kan starta, landa och flyga autonomt.
- Drönaren kan flyga en distans av 15km.
- Drönaren kan bära en last.
- Drönaren ska återvända till en förutbestämd position om 4G-täckning tappas.

1.3.2 Applikation

- I applikationen kunna följa drönaren i realtid via karta och videoström.
- I applikationen kunna övervaka rutten.

- I applikationen kunna läsa av batterinivå och uppskattad kvarvarande flygtid.
- I applikationen kunna se relevant logginformation.

1.4 Avgränsningar

För att underlätta projektet bestämdes att enbart en typ av drönare kommer undersökas då konstruktion och testning av olika drönartyper skulle ta för lång tid. För att kunna få bästa möjliga drönare för arbetet bestämdes det att ett quadplane skulle användas. Detta då denna drönartyp kommer leda till lägre energiåtgång än en quadcopter och därmed längre räckvidd, samtidigt som start och landning blir flexiblare än med ett flygplan utan VTOL-funktion[5].

Det bestämdes även att drönaren kommer starta och landa på områden avsedda för detta. Att starta och landa på godtycklig plats är en omfattande uppgift som egenhändigt skulle kunna uppta hela projekttiden. Det är dessutom rimligt att anta att sjukhusen har möjlighet att förbereda speciella ytor för start och landning då de redan har helikopter-plattor.

1.5 Etiska aspekter

Det finns flera positiva och negativa samhälleliga och etiska aspekter som bör tas i beaktande vid uppstart av ett projekt som innefattar både drönare och transporter. Genom att flyga drönare med en uppkopplad kamera är det lätt att stöta på problem gällande inkräktande på individers personliga sfär då personer lätt kan bli filmade utan deras tillåtelse [6]. Om en person kan identifieras utifrån foton eller video tagna av drönaren blir den fotograferande parten personuppgiftsansvarig för personuppgifter som fångats av kameran [7]. Detta problem togs även upp av svenska staten då det diskuterades huruvida drönarflygnng skulle klassas som någon typ av övervakning och därmed behöva speciella tillstånd för flygning. Beslutet som fattades var dock att det skulle vara tillåtet att flyga utan tillstånd [8], men bara för att det lagmässigt är inom ramarna betyder det inte att det nödvändigtvis är etiskt.

En annan aspekt som måste tas i beaktande är användandet av drönare för att transportera saker av både ekonomiskt och personligt värde. När något av stort värde transporteras måste det säkerställas att föremålet inte kan tappas, förstöras eller bli stulet. Detta

blir extra viktigt om något från sjukhus eller andra samhällskritiska funktioner [9] transporteras då det skulle kunna leda till stora dröjsmål eller problem om prover och liknande skulle försvinna.

Eftersom drönaren är tänkt att användas för leveransuppdrag inom sjukvården kan det antas att den kommer flyga över befolkade områden. Om den då skulle haverera kan detta leda till problem. Dels kan känslig last gå förlorad, men det kan även leda till mer direkta problem som kollision med människor eller fordon. Det är därför viktigt att förhålla sig till Transportstyrelsens regler kring flygning av drönare, särskilt vid flygning i eller nära en flygplats kontrollzon [10].

Om drönaren skulle ha förmågan att släppa föremål från höga höjder skulle detta kunna missbrukas då man med relativ anonymitet får möjlighet att transportera olagliga laster. Risken finns dessutom att drönaren skulle kunna användas för att skada andra genom att släppa någon typ av explosivt, giftigt eller på annat vis skadligt material på platser där många människor rör sig.

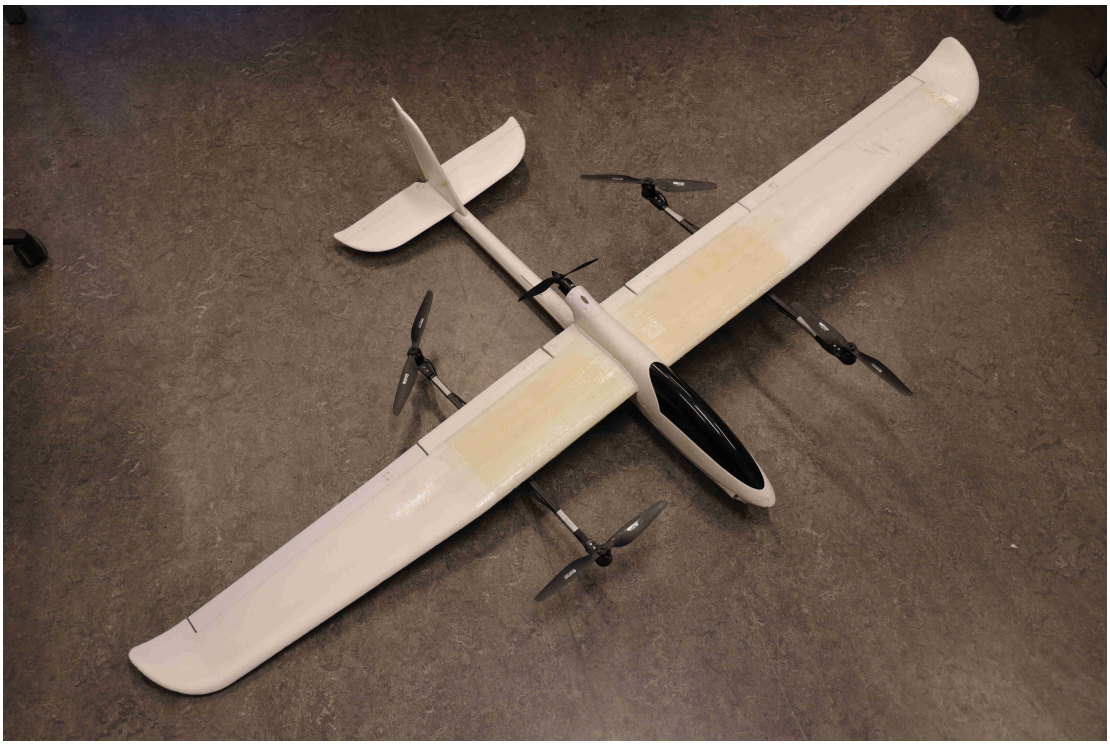
Problemen som skulle kunna lösas av projektet är dock ganska stora. Att kunna transportera gods snabbt och enkelt skulle kunna effektivisera och lösa problem inom till exempel sjukvården vilket skulle vara till stor samhällsnytta.

2 Metod

Projektet består av två centrala delar, en drönare och en applikation. Delen som rör drönaren är mest relaterad till hårdvara medan applikationen enbart relaterar till mjukvara. Denna uppdelning av kompetensområden har lett till tydlig uppdelning av projektet och utvecklingen av dess olika komponenter.

2.1 Drönare

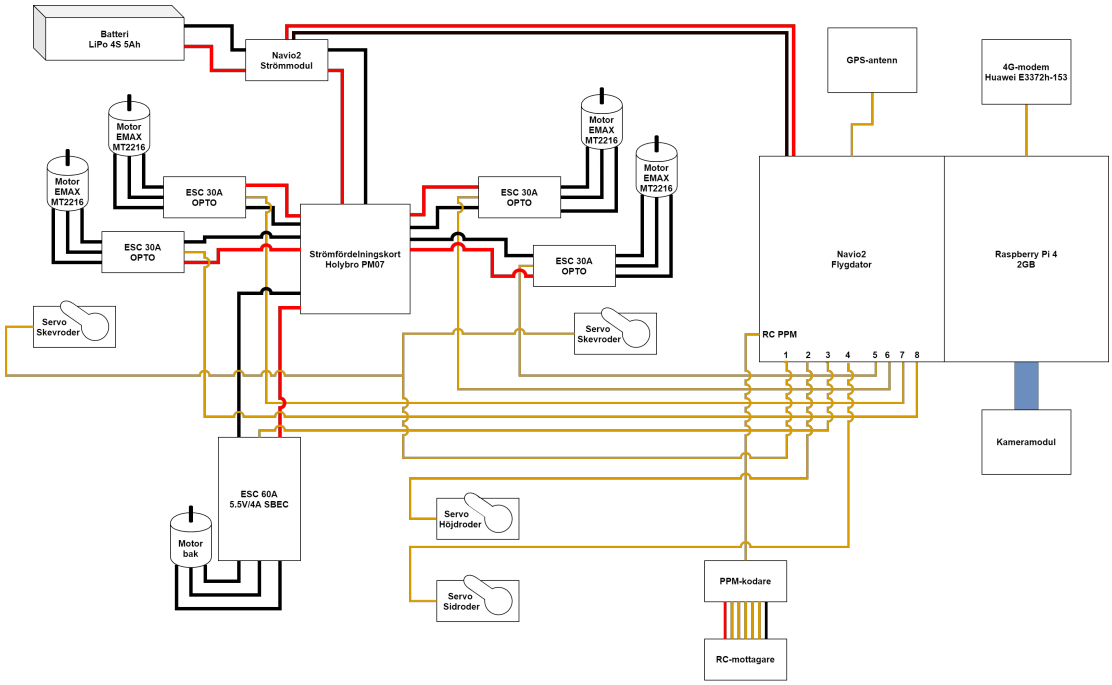
I projektet används som tidigare nämnt ett quadplane för att, i teorin, kombinera det bästa av två världar. Från föregående års projekt fanns det redan en färdig prototyp av ett quadplane med eftersökta egenskaper (figur 2.1) och ett vingspann på cirka $2\,\mathrm{m}$, vilket. Dock saknades en del essentiella delar för projektet.



Figur 2.1: Prototyp från föregående års projekt

2.1.1 Komponenter

För att få en överblick över vilka delar som kommer behövas sammanställdes i tidigt skede ett kopplingsschema (figur 2.2). Detta kopplingsschema beskriver hur samtliga elektriska delar ska kopplas samman.



Figur 2.2: Det fullständiga kopplingsschemat över drönaren med alla dess komponenter

Från drönarens flygdator, en Navio2 kombinerad med en Raspberry Pi, går det styrsignaler till alla servon och ESC:er som används för att styra drönaren. Denna kombinerade flygdatorenhet drar ström från batteriet genom en batterieleminator. Samma batteri förser även alla ESC:er med kraft så de kan driva motorerna. Dessutom finns en modul kopplad till detta för att möjliggöra traditionell RC-styrning av drönaren.

2.1.2 Radiostyrning

För att underlätta i utvecklingen och testningen av drönaren styrs den i början av projektet med klassisk radiostyrning. Radiostyrning är också användbart vid manuellt övertagande när autopilot testas på kortare avstånd. På grund av projektets mål med långdistansflygning kommer det varken vara möjligt eller önskvärt att endast använda radiostyrning för hela projektet.

2.1.3 Stötdämpare

För att minimera flygplanskroppens vibrationers påverkan på flygdatorn behövs någon form av stötdämpare. Utan den kommer mätdatan från flygdatorn bli mindre pålitlig.

Företaget som producerar flygdatorn Navio2 (Emlid) tillhandahåller en färdig design för en stötdämpare som passar ändamålet [11]. Denna skrivs ut med en 3D-skrivare och används för att montera flygdatorn i flygplanskroppen (figur 3.4).

2.1.4 Last

Lasten som i vårt scenario ska transporteras är mycket känslig och av stor betydelse och måste därför fraktas på ett sätt som minimerar risken för att den blir skadad, förstörd eller bortkommen. För att transportera lasten krävs en behållare som uppfyller en rad olika kriterier. Behållaren ska vara hållbar och robust men samtidigt vara av låg vikt för att dels ha minimal påverkan på drönarens flygegenskaper, men även för att inte begränsa tyngden på själva lasten för mycket. För att på enklast möjliga sätt uppnå dessa krav skapas behållaren från grunden genom att modelleras i CAD och därefter skrivas ut med en 3D-skrivare. Detta för att skapa en behållare med optimala mått med relativ låg vikt då den kommer bestå av plast [12]. Lastbehållaren placeras på planets undersida för att lasten ska kunna släppas på enklast möjliga sätt.

2.1.5 4G-uppkoppling

För anslutningen till 4G-nätet används ett USB-anslutet modem av modell *Huawei E3372h-153*. Detta främst då modemet presenterar sig som en vanlig ethernet-port och fungerar därmed utan vidare konfiguration i Linux [13], vilket är operativsystemet som körs på Raspberry Pi i drönaren. En nackdel är att modemet har inbyggd routingfunktion och gör addressöversättning vilket innebär att vår Raspberry Pi inte kan få en publikt addresserbar IP-adress. Detta problem löses med hjälp av VPN.

2.1.6 VPN

Då troligtvis varken drönare eller markstation kommer ha publikt adresserbara IP-adresser under normala omständigheter kommer trafiken mellan dessa behöva tunnlas. Detta realiseras genom en extern VPN-server som både drönare och markstartion ansluter till och kan prata med varandra via. Protokollet som används är Wireguard, vilket är ett modernt VPN-protokoll som inte kräver mycket prestanda, är enkelt att konfigurera och inte introducerar mycket fördröjning [14].

2.2 Applikation

Utvecklingen av applikationen består av två huvudsakliga uppgifter: hantera datan från drönaren och göra den tillgänglig för applikationen på ett säkert och smidigt sätt, samt utvecklingen av själva applikationen och dess komponenter.

2.2.1 Mavlink

Drönaren sköter som standard kommunikationen med markstationen via protokollet Mavlink [15] för vilket det finns flertalet färdiga bibliotek som används i det här projektet. Viss hänsyn måste tas i valet av dessa eftersom applikationen utvecklas för Android och biblioteken då måste stödja det. Eftersom applikationen enbart är tänkt som en lyssnare och inte ge kommandon räcker det dock med att tolka och förstå meddelanden.

2.2.2 Utveckling

Applikationen utvecklas för operativsystemet Android då det är relativt simpelt att utveckla och testa, samt att en mobilapplikation möjliggör ökad mobilitet för användaren gentemot ett datorbaserat program. Utvecklingsmiljön som används är Android Studio och källkoden till applikationen skrivs i Java. För att undvika kompatibilitetsproblem utvecklas applikationen mot Android version 5.0 (Lollipop) vilket är en äldre variant än det som återfinns på moderna enheter. Stora delar av applikationen använder sig av grundfunktionaliteten i Android men användningen av externa API:er är nödvändig. Kommunikationen med drönaren använder Dronefleets Mavlink API och för visualisering av positionsdata används Google Maps API.

3 Genomförande

För att projektet skulle lyckas krävdes det att de två stora delarna; drönarkonfigurationen och applikationen, blev genomförda separat, men också att de kunde kopplas ihop till en fullvärdig produkt.

3.1 Drönare

Utvecklingen av drönaren påbörjades med ett skal av en drönare från ett tidigare projektarbete. Därefter kunde de olika delarna så som flygdatorn, motorer, batterier och andra

nödvändiga delar installeras och testas. När drönaren färdigställts behövde den testas för att säkerställa att den uppnått de uppsatta kraven.

3.1.1 Flygdator

Initialt var planen för projektet att använda en Pixhawk 4 som flygdator. Det enskilt största argumentet för detta var att en sådan enhet redan fanns på plats redo att användas. Detta möjliggjorde att projektet skulle flyta på snabbare och därmed ha en drönare i luften betydligt snabbare. Pixhawk 4 är en välkänd flygdator som går att koppla ihop med en Raspberry Pi [16] för att uppfylla projektets kravspecifikation. Dock uppstod snabbt problem vid konfigureringen.

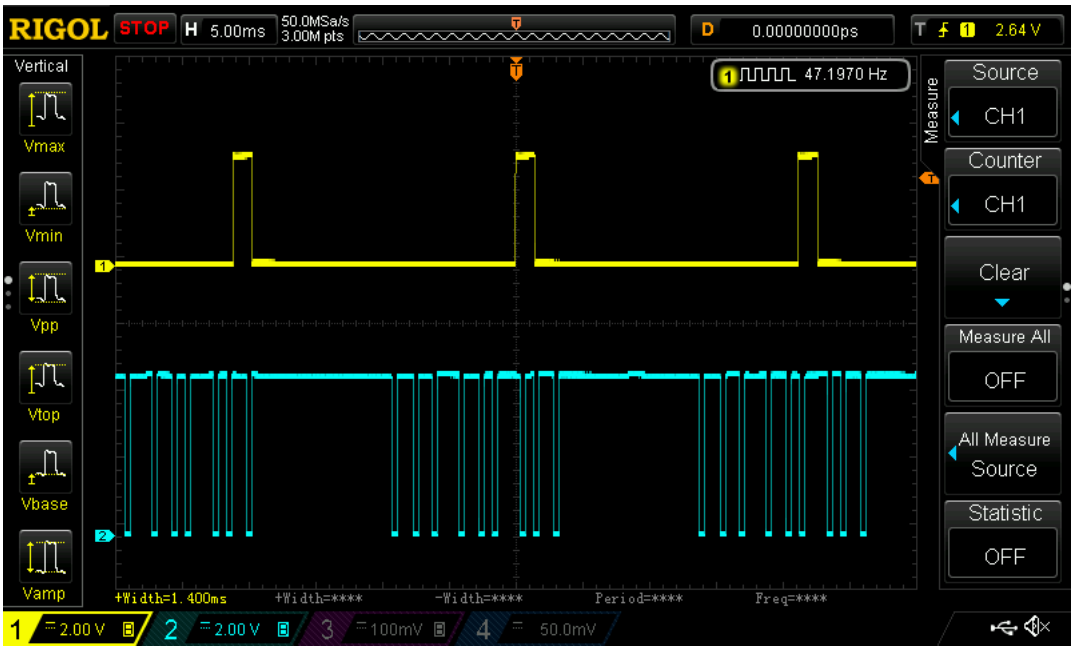
Först var enheten inte nåbar. När den kopplades till en dator för konfigurering och uppdatering av mjukvara gav den ingen respons. Efter fabriksåterställning av dess båda processorer började den kommunicera. Detta antogs bero på tidigare konfiguration i andra projekt. Vid mjukvaruuppdatering slutade enheten svara tills den fabriksåterställdes igen. Detta upprepade sig vid varje konfigureringsförsök.

Noterbart var att den aldrig lyckades hitta någon GPS-position. Felsökning av GPS-modul och dess kontakt med Pixhawken påbörjades. Först togs stiftlayouten för både GPS-modulen [17] och Pixhawken [18] fram och sammakopplades med en adapter för att underlätta åtkomst till varje stift. Spänningen från Pixhawken mättes till $0\,\mathrm{V}$ oavsett konfiguration. När GPS-modulen strömmatades externt startade den och gav som förväntat diagnostik via ljudsignaler, dock endast felmeddelanden som bekräftade att Pixhawken var obrukbar och beslut togs i samverkan med handledare att istället införskaffa en Navio2 som flygdator.

Navio2 är ett expansionskort till Raspberry Pi som använder den öppna flygdatorprogramvaran ArduPilot för styrning [19], [20]. ArduPilot har stöd för både autopilot och quadplane-läge, vilket gör det till ett bra alternativ för vårt syfte. För Navio2 tillgodoser tillverkaren en egen Linux-distribution för ändamålet som laddades på minneskortet för att sedan starta upp den kombinerade enheten. Detta skedde utan problem och nödvändiga programvaror installerades genom den inkluderade pakethanteraren.

3.1.2 Radiostyrning och PPM-kodare

Den radiomottagare som fanns tillgänglig var en modell med 7 kanalers PWM-styrning som är lämplig att ansluta direkt till antingen servomotorer eller ESC:er för direkt styrning, men tyvärr har flygdatorer i allmänhet (Navio2 inkluderat) inte stöd för att direkt ansluta 7 separata PWM-kanaler utan väntar sig istället en enskild signal av PPM-typ som innehåller samtliga kanaler. Lyckligtvis är detta ett vanligt problem med en relativt enkel lösning, en så kallad PPM-kodare. I autopilotmjukvaran Ardupilot ingår ett program för att koda om upp till 8 PWM-signaler till en PPM-signal (figur 3.1). Detta program kan köras på den högst vanliga mikroprocessorn Atmel ATmega328P [21], vilken råkar vara densamma som finns på en Arduino Pro Mini [22] som både fanns tillgänglig och var liten nog att få plats inuti drönaren. Den färdigkompilerade programfilen programmerades till mikroprocessorn och fungerade bra med alla kanaler utom kanal 2. Efter felsökning visade det sig att programmet inte var kompatibelt med den vanliga Arduino-bootloadern eftersom det använder en av pinnarna som bootloadern reserverat för seriekommunikation som ingång för just kanal 2. Genom att använda en ISP-programmerare (i vårt fall användes en annan Arduino för detta) och skriva programmet direkt till mikroprocessorn utan att använda bootloadern fungerade även kanal 2 som väntat.



Figur 3.1: Exempel på PWM- respektive PPM-signal. Kanal 1: Enkel kanal PWM-signal från radiomottagare. Kanal 2: PPM-signal innehållandes 8 sådana PWM-signaler.

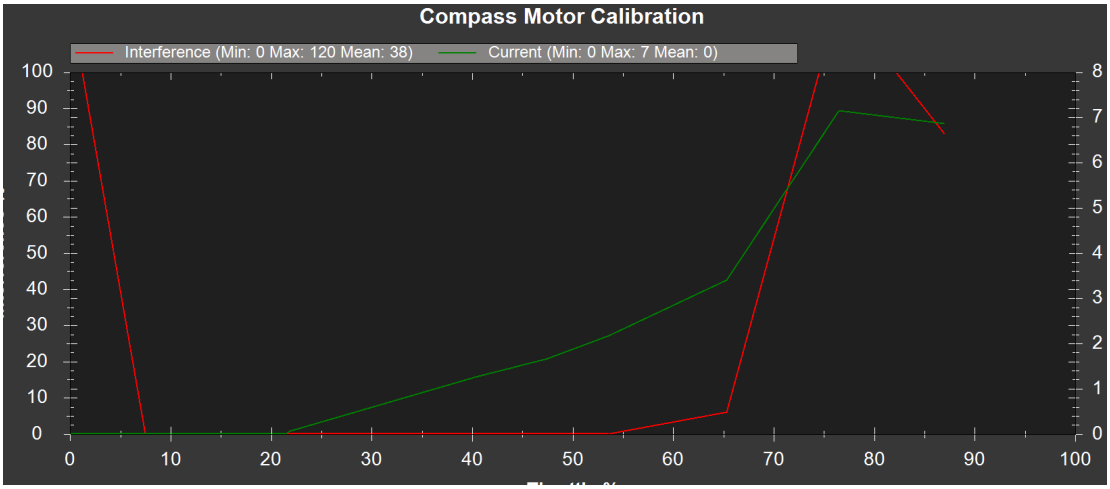
3.1.3 Quadcopter

Inledningsvis konstruerades en quadcopter i flygplanskroppen. Samtliga servomotorer samt motorn för framåtdrift utelämnades. Trots den begränsade mängden komponenter blev platsbrist ett problem. Därför placerades de 4 ESC:erna, som ansvarar för de vertikala motorerna, på utsidan av flygplanskroppen. Detta säkerställer även att de inte riskerar överhettning, vilket var en potentiell risk då flertalet komponenter monterats tätt inpå varandra i flygplanskroppen.

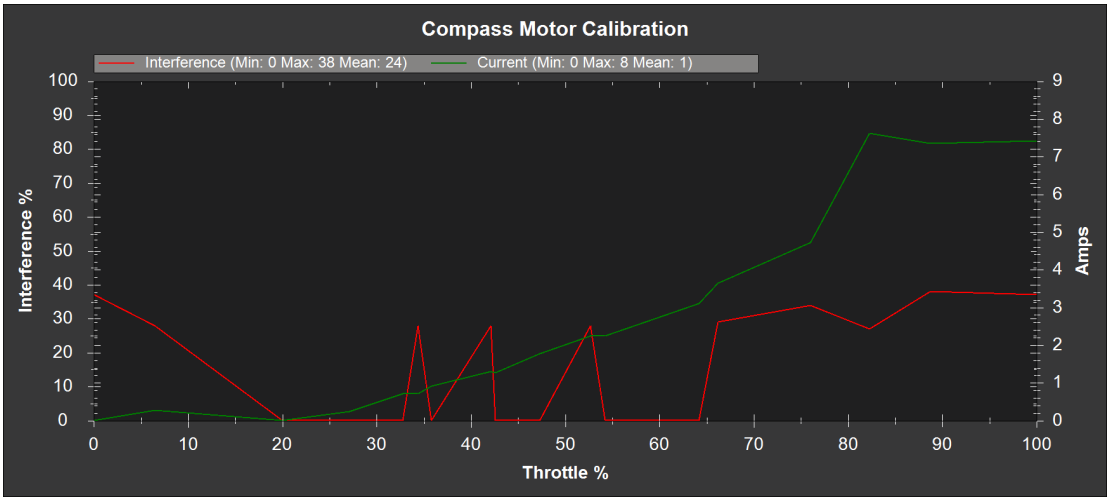
Efter komponenter monterats inleddes provflygningen och drönaren flög väl med manuell styrning. Därefter testades autoläget, loiter. I läget loiter ska drönaren hålla sig på ungefär samma position i alla tre dimensioner och behålla sin riktning om inga kommandon ges från operatören [23]. När loiter aktiverades började drönaren kraftigt svänga fram och tillbaka och var omöjlig att kontrollera.

3.1.4 Felsökning av stabilitetsproblem

Efter litteraturstudier om orsaken bakom liknande instabilitet genomfördes ett CompassMot-test [24] för att undersöka om motorerna stör ut magnetometern. Testet visade att så delvis var fallet. Motorerna i sig var inget problem men batterieleminatorn som strömförsörjer flygdatorn gav upphov till väldiga störningar när motorerna snurrade (figur 3.2). Genom att placera batterieleminatorn längre bort från sensorn kunde mängden störningar minskas drastiskt, vilket syntes tydligt i det efterföljande CompassMot-testet (figur 3.3).



Figur 3.2: Mätning före flytt av batterieleminator. Störningarna är påtagliga.

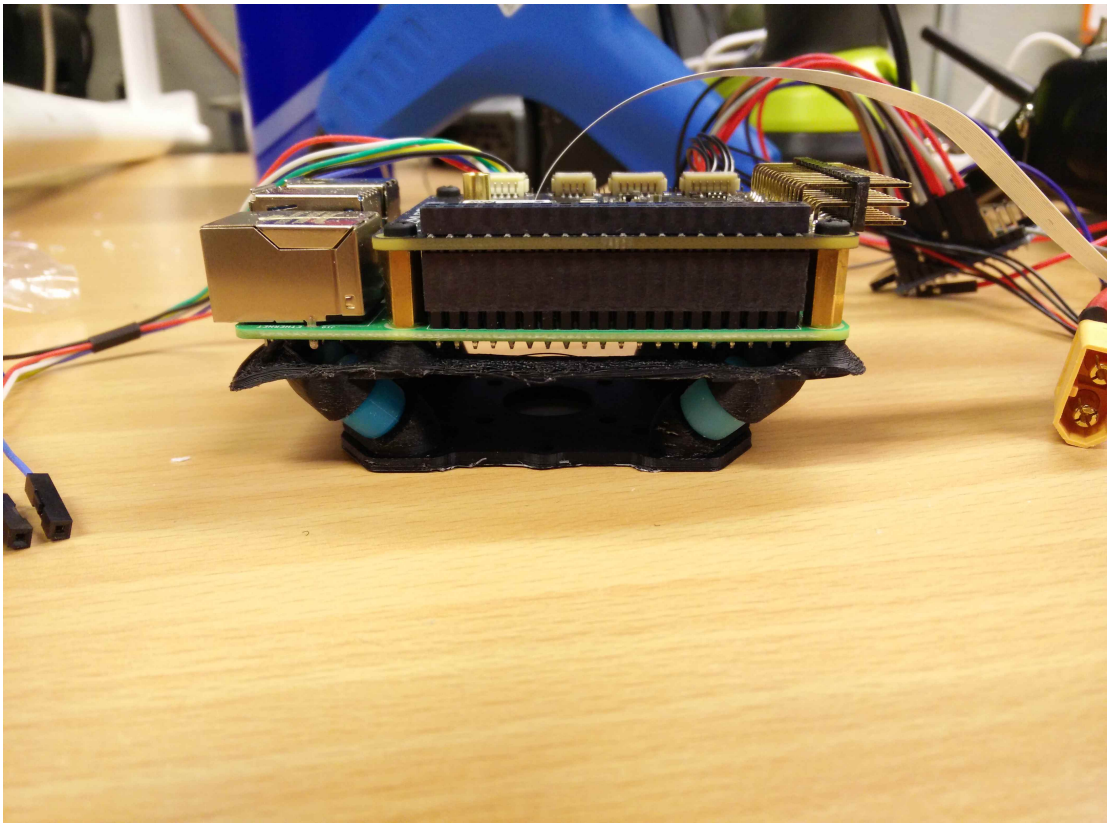


Figur 3.3: Mätning efter flytt av batterieleminator. Störningarna är betydligt mer lindriga.

Trots de minskade störningarna förbättrades inte stabiliteten hos drönaren. Dåligt väder förhindrade ytterligare tester och felsökning så problemet bortsågs från och återkom aldrig. Orsaken var troligtvis dåligt anpassade regulatorparametrar, vilka berodde på att flygdatorn programmerats till quadcopterläge. Detta läge var sannolikt inte anpassat för att hantera något med stora vingar som kraftigt påverkar drönarens manövreringsförmåga och därmed systemets dynamik. Nästa flygning genomfördes med quadplane-läget inställt och någon instabilitet kunde inte längre observeras.

3.1.5 Quadplane

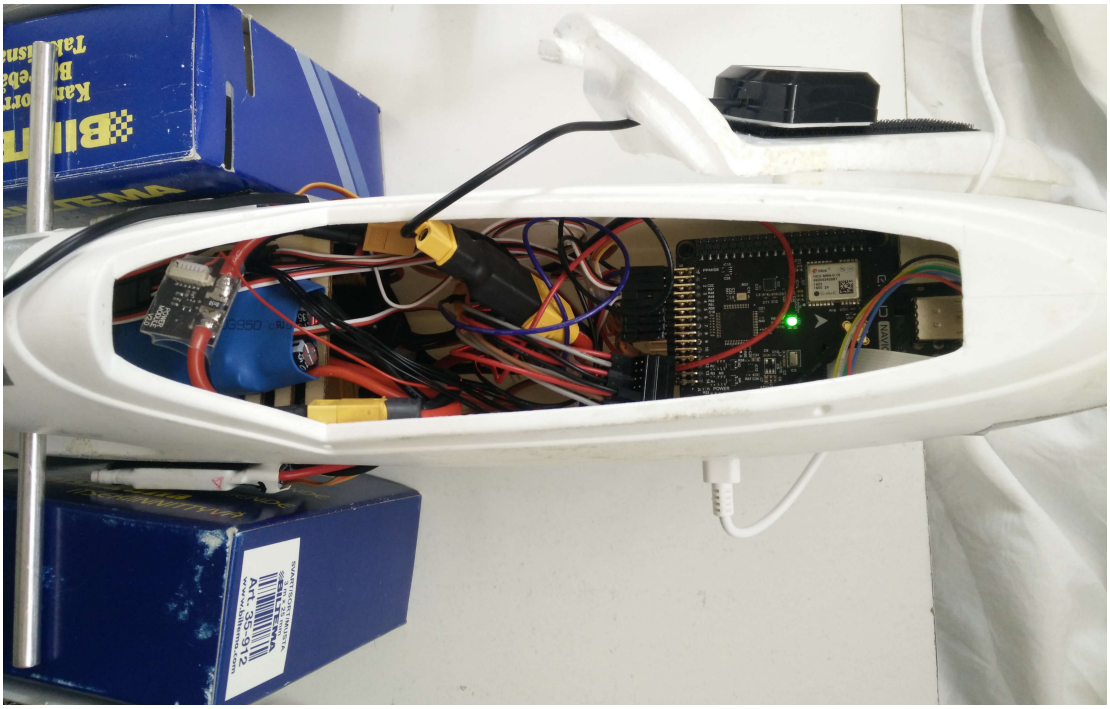
Inför flygandet av den slutgiltiga drönaren monterades alla komponenter fast. Flygdatorn skruvades fast i flygplanskroppen via ett antivibrationsfäste som tillverkades med 3D-skrivare enligt Emlids egna design [11] (figur 3.4).



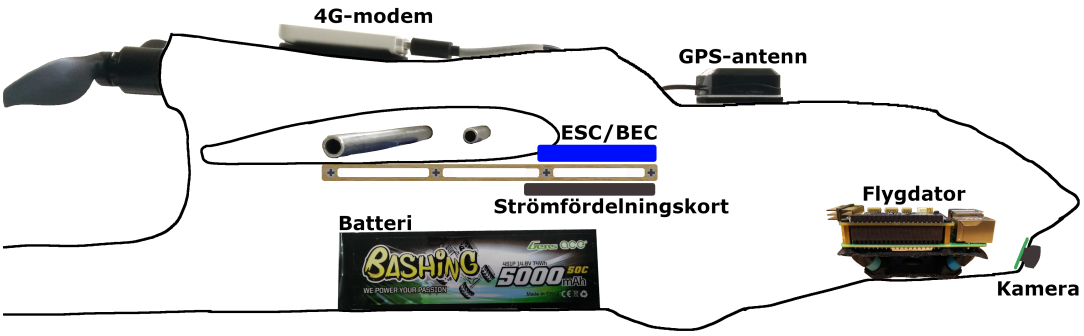
Figur 3.4: Navio2 monterad på stötdämpare

I detta skede monterades även kameramodulen fast på ett beständigt vis i planets nos,

strömfördelningskortet skruvades fast och batteriet begåvades med kardborreband för att sitta fast bra men samtidigt vara möjligt att plocka ur planet för att laddas med relativ lätthet. Motorn för framåtdrift anslöts och likaså de fyra servomotorerna som kontrollerar skev-, höjd- respektive sidoroder. På grund av platsbrist (figur 3.5) i flygplanskroppen monterades det USB-anslutna 4G-modemet på ovansidan istället för inuti (figur 3.6). Förhoppningen var att detta även skulle ge bättre prestanda eftersom avståndet till komponenter som kan generera störningar blir större. Figur 3.7 visar den färdigbyggda drönarens slutliga utseende.



Figur 3.5: Insidan av framdelen av flygplanskroppen



Figur 3.6: Placering av de olika komponenterna inuti flygplanskroppen

Figur 3.7: Den färdigbyggda drönaren

3.1.6 Kalibrering

För att flygdatorn skulle kunna styra servon och ESC:er behövde en kalibreringsprocess genomföras. De fyra ESC:er som driver vertikalmotorerna är av kalibreringsfri modell så detta steg var ej nödvändigt i det tidigare quadcopterläget. Den femte ESC:n, som

hanterar den framåtdrivande motorn, behövde kalibreras enligt Ardupilots kalibreringsprocess [25]. Servomotorerna behövde i sin tur kalibreras så att mittpunken svarar mot att motsvarande roder är helt rakt och så att motorns omfång är anpassat till hur mycket rodret ska kunna röra sig.

3.1.7 Felsäkerhet

För att undvika att drönaren flyger utanför områden med täckning och blir onåbar konfigurerades en felsäkerhetsfunktion som ser till att drönaren automatiskt flyger tillbaka till startpunkten om den tappar kommunikation med markstationen, vilket är vad som skulle hända om drönaren flyger utanför område där mobilnätet har täckning. Drönaren ska då först flyga tillbaka till startpunkten och där genomföra en övergång till VTOL-läge och landa. För att testa detta bryts kopplingen från markstationen istället för att flyga iväg drönaren långt bort eftersom det då fortfarande är möjligt att ta manuell kontroll med direkt radiostyrning om något skulle gå fel.

3.1.8 Provflygning

För att verifiera att drönaren uppfyller kravspecifikationen behövde en provflygning genomföras. För att minimera risken för krasch delades provflygningen upp i flera delmoment som gav möjligheten av verifiera olika delsystem under mer kontrollerade omständigheter.

Det första delmomentet som testades var helt manuell kontroll i VTOL-läge, alltså som en vanlig quadcopter med manuell kontroll. I ArduPilot kallas detta läge QSTABILIZE. Detta delmoment visar om VTOL-systemets alla motorer och propellrar fungerar som de ska och testar dessutom flygdatorns grundfunktionalitet, gyro och accelerometer [26].

Nästa delmoment som testades var höjdhållningsläget QHOVER. Detta bygger på QSTABILIZE, men lägger till automatisk låsning av höjd baserat på flygdatorns barometer [27].

Det sista delmomentet i VTOL-läge är positions- och riktningshållning i alla dimensioner, QLOITER. Detta läge introducerar, utöver de två tidigare testade lägena, nyttjande av GPS- och kompassdata för att fixera drönarens position och riktning i horisontalplanet och testar drönarens förmåga att hålla sig stabil i en position utan att operatören behöver

göra några manuella korrigeringar [28].

Därefter testades flygplansläget. För att börja flyga som ett flygplan behöver drönaren först starta i VTOL-läge och sedan genomföra en övergång till flygplansläge. FBWA är ett läge som tillåter i huvudsak manuell kontroll i flygplansläge men med stabiliseringshjälp från flygdatorn [29]. Genom att starta i QLOITER-läget och övergå till FBWA gör drönaren självmant en sådan övergång och VTOL-motorerna fortsätter ge lyftkraft tills planet nått tillräckligt hög hastighet framåt för att hålla sig i luften utan dem [30]. Detta test verifierar både själva övergången och att samtliga styrkontroller för flygplansläget fungerar.

I det sista delmomentet testades positionshållningsläget för flygplan, LOITER. I detta läge cirkulerar drönaren kring en bestämd punkt i flygplansläge. Detta testar att flygdatorn klarar av att hålla planet under kontroll utan manuell styrning [31].

Det som återstår är att genomföra det kompletta testet. Ett autonomt uppdrag programmeras därför in i drönaren enligt följande:

1. Starta i VTOL-läge
2. Genomför övergång till flygplansläge
3. Flyg till ett riktmärke en bit bort
4. Flyg tillbaka till startpunkten
5. Genomför övergång till VTOL-läge
6. Landa

För ett lyckat test ska drönaren helt autonomt utföra hela uppdraget.

3.1.9 Last

På grund av tidsbrist hann inte en mekanism för att automatiskt lämna av last utvecklas.

3.2 Applikation

Applikationen kommer bestå av flera olika komponenter och eftersom de olika komponenterna är relativt fristående kunde arbetet enkelt delas upp. Olika uppgifter delegerades till olika gruppmedlemmar som kunde utföra dessa. De kopplades sedan samman för att skapa den färdiga applikationen.

3.2.1 Server

I tidigt skede blev det tydligt att en serverdator skulle behövas för diverse syften, inte minst för VPN-server. Kraven på denna var att den skulle ha en publik IP-adress så att den kan nås från hela internet och att den skulle ha nog med prestanda för att inte bli en flaskhals i systemet. En sådan server kopplades upp hemma hos en gruppmedlem och kommer hädanefter benämnas “servern”.

3.2.2 VPN

Som tidigare nämnts behövde trafiken mellan drönare och markstation tunnlas för att garantera kommunikation. Här användes det relativt nya VPN-protokollet Wireguard. Wireguard installerades på server, drönare och markstation och konfigurerades så att både drönare och markstation kunde ansluta till servern. Med denna uppsättning kunde dessa två sedan kommunicera med varandra oberoende av fysiskt avstånd och utan krav på att de själva skulle ha publikt adresserbara IP-adresser så länge båda hade internetuppkoppling.

Då anslutningen i drönaren måste hårdkodas till en adress användes samma VPN-konfiguration för alla datorer som kan agera markstation. Därmed kan i princip vilken dator som helst användas som markstation så länge den har rätt VPN-konfiguration. Dock kommer det med nackdelen att bara en användare kan vara ansluten direkt till drönaren åt gången.

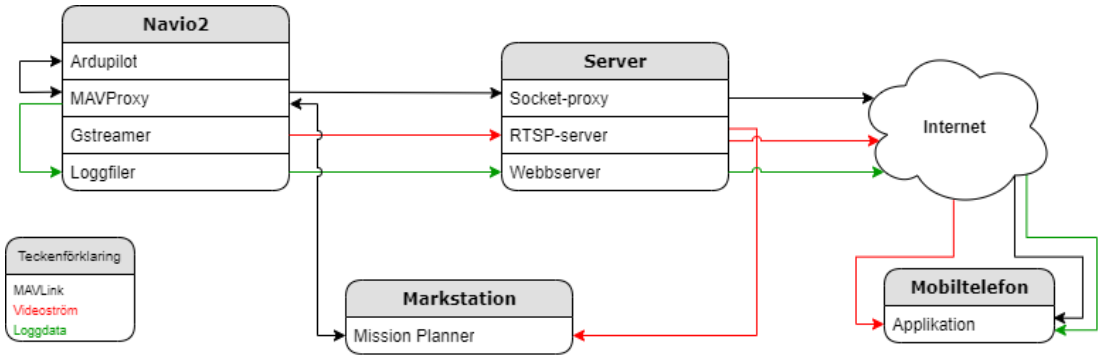
3.2.3 Backend

På grund av de förändrade förutsättningar utökningen av projektet med Android-applikationen innebar krävdes en del förändringar i den underliggande infrastrukturen. Ursprungligen var kommunikationen mellan drönare och markstation en relativt simpel point-to-point-anslutning över VPN på samma subnät där videoström och övrig kommu-

nikation gick direkt till en förutbestämd IP-adress. Med multipla användare via var sin applikation var detta inte längre gångbart och behövde anpassas (figur 3.8).

Videoströmmen skickades direkt från drönare till markstation via GStreamer för dess låga fördröjning. GStreamer klarar dock inte flera mottagare och dessutom är telefonerna med applikationerna nästan garanterat inte publikt adresserbara. Applikationerna behöver således ansluta till drönaren. Denna är dock inte heller inte adresserbar utan VPN och att få applikationen att ansluta genom VPN tillför onödig komplexitet. Istället utnyttjades ett bibliotek för att bygga en RTSP-server baserad på GStreamer [32]. Med detta implementerades ett program som tar emot en ström från drönaren och sedan låter användare ansluta. Då servern i projektet är publikt adresserbar kan applikationen ansluta direkt till denna utan VPN. Av säkerhetsskäl krävs användarnamn och lösenord för att få åtkomst till videoströmmen. I applikationen kan sedan valfritt API/SDK som kan ansluta till en RTSP-server användas.

För att få ut data från drönaren, såsom GPS-koordinater, hastighet, höjd och liknande, behövdes en mer avancerad lösning. Samma adresseringsproblematik som fanns vid videoströmningen återfanns även här. För att lösa detta användes MAVProxy. Som namnet antyder är detta en GCS som även kan agera proxy åt Mavlink-signaler. Då Ardupilot har begränsningen att endast kunna ansluta till en markstation via IP behövde MAVProxy köras. Navio2 anslöts till MAVProxy på dess Raspberry Pi som sedan skickade vidare till både markstation, ansluten via VPN, och servern. På servern kördes en socketproxyserver med syftet att ta den inkommande UDP-strömmen från MAVProxy och vidarebefordra till godtyckligt antal TCP-anslutningar så applikationen kan ansluta och få tillgång till datan. Detta realiserades genom att UDP-strömmen skrevs ner till en fil i filsystemet med verktyget socat (SOcket CAT) [33]. Denna fil gjordes sedan tillgänglig över TCP för flera anslutna applikationer. Även här användes socat. Eftersom det är en strikt envägskommunikation garanteras att applikationen enbart blir en lyssnare och inte kan ge uppdrag till drönaren. Då anslutningen mellan drönare och applikation saknar autentisering är det också, ur ett säkerhetsperspektiv, önskvärt att utomstående ej kan ge drönaren uppdrag.



Figur 3.8: Komplett figur över den uppdaterade infrastrukturen som applikationen kräver.

För att göra loggdatan från drönaren tillgänglig för applikationen krävdes ytterligare infrastruktur. Drönaren sorterar och sparar ner loggarna i sitt filsystem. Mappen med alla loggar kopieras över till servern via Rsync. Detta görs automatiskt varje natt eller på begäran från applikationen. Loggarna gjordes sedan tillgängliga via en webbserver som applikationen kan ansluta till via HTTPS och hämta valda loggar. Webbservern kräver autentisering för att förhindra obehörig åtkomst. På servern ligger loggarna lagrade som råa loggfiler tills en begäran kommer. Först då tolkas loggfilerna och skickas i JSON-format till applikationen.

3.2.4 Mavlink

För att implementera kommunikationen mellan drönare och applikation behövdes ett externt bibliotek. Till detta fanns tre huvudkandidater som alla utlovade den eftersökta funktionaliteten. Dessa tre var MAVSDK-Java [34], DroneKit [35] samt dronefleet [36].

Efter testning föll valet på dronefleet. MAVSDK-Java är enbart i beta-fas [37] och anpassat för PX4 [38]. Kompatibilitet med ArduPilot kan därför inte garanteras. DroneKit saknar dokumentation och är inte uppdaterad på mycket länge och kan därför i praktiken klassificeras som död. Dronefleet ger lätt åtkomst till data från drönaren och kan enkelt anpassas till att sortera ut enbart efterfrågad data. Detta är mycket önskvärt då inte all data från drönaren används i applikationen.

Dronefleet kombinerades med ett API för reaktiv programmering [39] för att kunna uppdatera applikationen allt eftersom ny data mottages från drönaren. Detta gör applika-

tionen flexibel för förändringar och minimerar kodbasen.

3.2.5 Karta och ruttövervakning

Att se drönarens position på en karta och följa själva flygningen är en av de mest centrala komponenterna i applikationen. För att presentera denna information valdes Google Maps API då detta fungerar bra med Android och inte kräver alltför mycket handpåläggning. En API-nyckel krävs för anslutningen till Google Maps servrar vilket Google erbjuder kostnadsfritt. I applikationen initieras en färdig komponent för kartan som kan användas till att markera positioner och rutter.

För att på enklast möjliga sätt kunna byta mellan olika vyer initieras kartan i form av ett Fragment som ersätter huvudinnehållet på den aktiva sidan.

3.2.6 Videoström

För att uppnå så låg fördröjning på videoströmmen som möjligt var planen ursprungligen att använda ett Gstreamer-API i vår applikation för att koppla till och spela upp videoströmmen. Detta visade sig dock vara onödigt komplicerat eftersom API:et inte var skrivet i samma programspråk som applikationen. Istället användes Androids egna videovy som klarar av att spela upp en videoström över RTSP. Då applikationen inte ska användas till att styra drönaren, utan endast övervaka den, är låg fördröjning inte lika viktigt och den simpla videovyn är då tillräcklig.

På samma sätt som för kartan används ett Fragment för att starta videovyn som i sin tur är en Activity som innehåller designen. Detta gör att videon körs på huvud-Activityn i applikationen och logiken för denna täcker även videon. Detta medförde problemet att om en visuell uppdatering skedde, som exempelvis när skärmen roteras från stående till liggande, laddades RTSP strömmen om vilket tar tid. Ett undantag lades in så dessa uppdateringar inte sker så att videoströmmen kontinuerligt visas.

3.2.7 Batterinivå och flygtid

För att en användare enkelt ska kunna övervaka sin drönare och uppdragen som utförs krävs det att information, såsom batterinivå och flygtid, hämtas via dronefleet.

Genom att ha ett en metod som prenumererar på ett meddelande som skickas via

dronefleet kan metoden uppdatera ett värde i applikationen. Detta leder till att användaren alltid kan få en klar bild på hur länge drönaren har flugit eller hur länge den kan fortsätta flyga baserat på dess batterinivå. Detta kan utvecklas ytterligare och innefatta fler egenskaper beroende på vad som är viktigt för användaren att veta.

Flygtiden beräknas med hjälp av drönarens hjärtslag. Varje sekund skickar den ett meddelande med allmän status och för verifiering av kontakt. Från hjärtslagen går det bland annat att läsa ut om drönaren är aktiv eller i viloläge. Om tidpunkten när drönaren byter status sparas är det lätt att beräkna hur länge den varit aktiv. Det är också vad som görs i applikationen. Tiden fortsätter räknas till dess att drönarens status ändras tillbaka. Eftersom statusen vid landning kan skifta mellan att vara aktiv och i viloläge precis när den slår i marken lades ytterligare ett fält till. Där visas senaste flygning som varade längre än 10 sekunder. Utan detta finns det risk att datan om flygtid skrivs över vid landning av drönaren.

3.2.8 Logginformation

Enkel tillgång och tydlig presentation av tidigare flygningar ansågs vara en önskvärd funktion. Layouten för aktiviteten är en rullbar vy. Syftet med layouten är att kunna presentera tidigare flygningar i fallande eller stigande kronologisk ordning i en enda kolumn. Användaren kan då enkelt bläddra bland de tidigare flygningarna. När applikationen startar är layouten tom, men fylls under körning på med objekt som representerar varje enskild flygning. Då ett objekt väljs presenteras information om den valda flygningen för användaren.

För att uppnå detta måste först en lista med tillgängliga loggar hämtas från servern. Innan det kan ske måste denna lista uppdateras. Listan hämtas från servern med ett HTTPS-request som besvaras med en lista över tillgängliga loggfiler i JSON-format med information om datum och flygningsnummer (figur 3.9).

För att lättare hantera informationen i applikationen skapades en egen klass som konstrueras med strängen från servern. Denna klass innehåller datum, flygningsnummer, och URL-adressen till flygningens loggdata.

Efter att ett objekt skapats för varje tillgänglig logg sorteras dessa i kronologisk ordning. Flygningarna är nu redo att presenteras i applikationen för användaren. Aktivitetens

layout refereras i koden för att sedan fyllas med separata knappar som genereras för varje enskild flygning (figur 3.10).

```
0:    "2020-04-09/flight1"
1:    "2020-04-09/flight2"
2:    "2020-04-14/flight1"
3:    "2020-04-14/flight10"
4:    "2020-04-14/flight11"
5:    "2020-04-14/flight2"
6:    "2020-04-14/flight3"
7:    "2020-04-14/flight4"
8:    "2020-04-14/flight5"
9:    "2020-04-14/flight6"
10:   "2020-04-14/flight7"
11:   "2020-04-14/flight8"
12:   "2020-04-14/flight9"
13:   "2020-04-22/flight1"
14:   "2020-04-22/flight2"
```

Figur 3.9: Lista med strängar som representerar tillgängliga loggar.

Figur 3.10: Rullbar vy där varje knapp representerar en flygning.

När användaren trycker på en av knapparna skapas en ny aktivitet. Objektet som stämmer överens med knappen skickas med till den nya aktiviteten. I den görs en ny HTTPS-request till dess URL varpå loggen skickas som svar i JSON-format.

Loggen innehåller information om tid, position, hastighet, med mera. I den här delen av applikationen är det enbart relevant med drönarens koordinater vid varje tidpunkt under flygningen, samt tidpunkten själv. Med denna information kan vi, med hjälp av Googles API, plotta flygningens rutt på en karta samt räkna ut och presentera flygtiden.

4 Resultat

En autonom drönare som även klarar manuell styrning via 4G samt vertikal start och landning har tagits fram. Dessutom har en applikation för övervakning av drönaren utvecklats.

4.1 Drönare

I avsnitt 1.3.1 sattes följande kravspecifikation upp för drönaren:

- Drönaren kan styras och övervakas via 4G.
- Drönaren kan starta och landa vertikalt (VTOL).
- Drönaren kan flyga horisontellt.
- Drönaren kan starta, landa och flyga autonomt.
- Drönaren kan flyga en distans av 15km.
- Drönaren kan bära en last.
- Drönaren ska återvända till en förutbestämd position om 4G-täckning tappas.

Drönaren uppfyller 6 av de 7 kriterierna. Den är kapabel att flyga både som quadcopter och flygplan, och klarar även övergången mellan dessa. Kombinationen av dessa faktorer innebär att drönaren klarar att flyga som flygplan men starta och landa vertikalt. Det går också att programmera uppdrag till den som utförs autonomt. I händelse av tappad 4G-koppling flyger drönaren tillbaka till en tidigare bestämd punkt. Drönaren klarar även att bära en last som väger upp till $500\,\mathrm{g}$. Den klarar dock inte att flyga kravspecifikationens önskade $15\,\mathrm{km}$.

4.1.1 Flygprestanda

Provflygningarna visade att drönaren kan genomföra samtliga delmoment. I VTOL-läge är stabiliteten över lag väldigt bra, men det märks att drönaren inte är specialanpassad för quadcopterläge. Motorerna arbetar på ca $70\,\%$ av maxkapacitet bara för att hålla drönaren i luften. Detta blir snabbt märkbart när man försöker rotera drönaren kring vertikalaxeln, vilket i en quadcopter görs genom att störa kraftbalansen och låta ett motorpar (som har samma rotationsriktning) göra större delen av arbetet och därmed skapa ett nettomoment som får drönaren att rotera. Eftersom två motorer inte har lyftkraft nog att hålla drönaren i luften måste alla motorer hjälpas åt och två av dem kommer då motverka rotationen man vill åstadkomma. Detta är inte något stort problem eftersom

drönaren är tänkt att i huvudsak flyga som ett flygplan där samma problematik inte finns.

I flygplansläge är drönaren relativt tungrodd, men givet nog med utrymme och tillräcklig hastighet klarar den att flyga framåt, upp, ner och svänga. Då ingen i gruppen hade någon tidigare erfarenhet av radiostyrda flygplan är det svårt att avgöra hur den står sig i jämförelse med ett radiostyrt flygplan i samma storleksklass men som saknar VTOL-funktionen som både ökar vikten och försämrar aerodynamiken.

Det enskilt största problemet med vår drönare är hur känslig den är för väder och vind. Det finns inget som helst skydd för regn och redan vid måttlig vindstyrka får drönaren stora problem med att hålla position och kurs eftersom dess stora vingar utgör ett rejält vindfång. Detta har begränsat möjligheten att testa och finjustera parametrar. Det var relativt få dagar som väderförhållanden tillät flygning över huvud taget och på grund av att det bara fanns ett enda batteri som behövde laddas mellan varje flygtur kunde inte heller dessa dagar utnyttjas till fullo.

4.1.2 Felsäkerhet

Felsäkerhetsfunktionen fungerade som planerat. Om kommunikationen med markstationen bröts övergick drönaren till ett läge där den automatiskt återvände till startpunkten och genomförde där övergång till VTOL-läge samt landning och desarmering av motorer.

4.1.3 Räckvidd och batteriprestanda

I vårt räckviddstest flög drönaren 22 varv i en cirkel med radie 30 meter och förbrukade då $40\,\%$ av batterikapaciteten. Denna cirkel var väldigt snäv och det märktes att planet kämpade ganska hårt för att bibehålla sin höjd. Flygloggar visade att planets i snitt var vinklat 30 grader in mot cirkelns mitt, vilket innebär att endast $\cos(30) = 87\,\%$ av vingarnas lyftkraft var riktad uppåt. Resten var riktad i sidled och gick därmed förlorad jämfört med om planet flugit helt rakt. Därför antas en väldigt förenklad modell där detta innebär att räckvidden egentligen är $1/\cos 30 = 15\,\%$ större. Detta ger en total uppskattad räckvid på:

$$\frac{22[\text{varv}] \cdot 2\pi \cdot 30[\text{radie, meter}]}{0.4[\text{batteriförbrukning}] \cdot \cos 30[\text{vinkel}]} \approx 12\,000[\text{meter}]$$

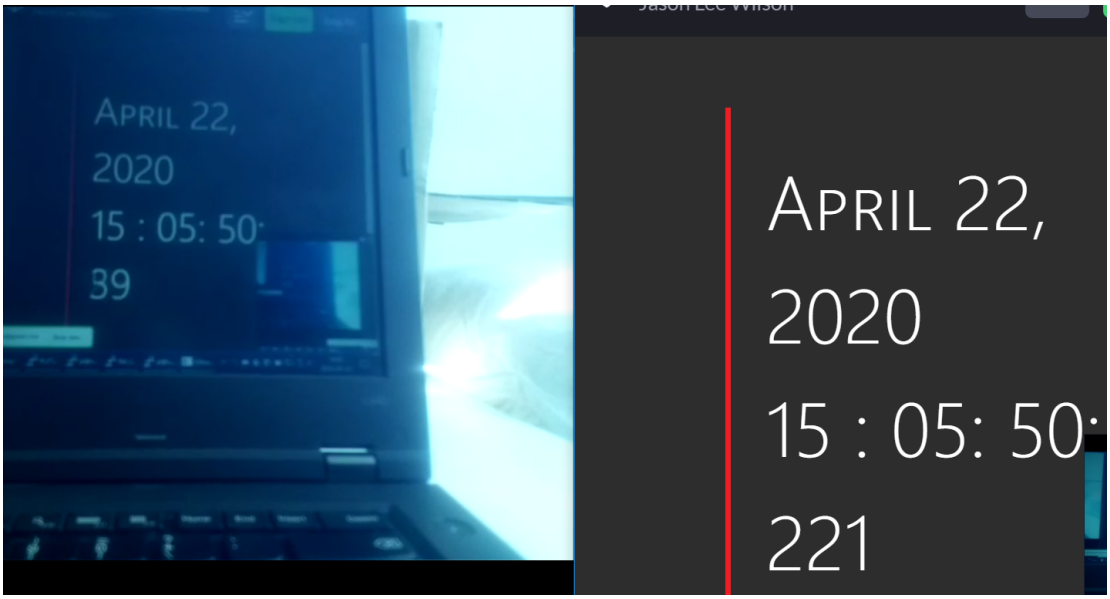
Drönarens räckvid borde alltså vara ungefär $12\,\mathrm{km}$, vilket innebär att den inte uppnår målet om $15\,\mathrm{km}$ från kravspecifikationen, men det är ändå med marginal nog för att flyga $7\,\mathrm{km}$ mellan Östra sjukhuset och Sahlgrenska sjukhuset enkel väg.

I VTOL-läge räcker batteriet till att hovra i ungefär 15 minuter, men under ett vanligt uppdrag är det inte tänkt att drönaren ska behöva mer än som absolut mest en minut i VTOL-läge för både start och landning. Inte ens när höga träd tvingar att övergång till och från flygplansläge vid en hög höjd borde så mycket som en minut krävas för start och landning.

4.1.4 Fördröjning över 4G

Det märktes ingen större skillnad mellan styrning över 4G och direkt radiostyrning. Drönaren är så pass stor, tung och långsam i manövrarna att de ca $50\,\mathrm{ms}$ som införts av hela kedjan med VPN och 4G inte blev särskilt märkbara.

Video var inte lika imponerande. Genom att peka kameran mot en klocka uppmättes videoströmmens fördröjning (figur 4.1). Testet visade att videoströmmen låg ungefär $180\,\mathrm{ms}$ efter verkligheten. Det visade sig att Raspberry Pi-kameran som användes inte var det bästa valet eftersom den införde en fördröjning på ungefär $150\,\mathrm{ms}$ [40]. I och med att vårt mätvärde även inkluderade en 4G-länk får vårt resultat anses vara acceptabelt, även om det inte är riktigt levde upp till förhoppningarna.



Figur 4.1: Mätresultat för videofördröjning

Det är trots fördröjningar möjligt att manuellt flyga drönaren utan att titta på något annat än videoströmmen och använda handkontrollen, dvs att all styrning sker över 4G-nätet. Däremot är det betydligt svårare att göra precisionsmanövrar jämfört med vanlig radiokontroll och drönaren i direkt siktlinje eftersom fördröjningen gör det väldigt enkelt att överkompensera.

4.2 Applikation

I avsnitt 1.3.2 sattes följande kravspecifikation upp för applikationen:

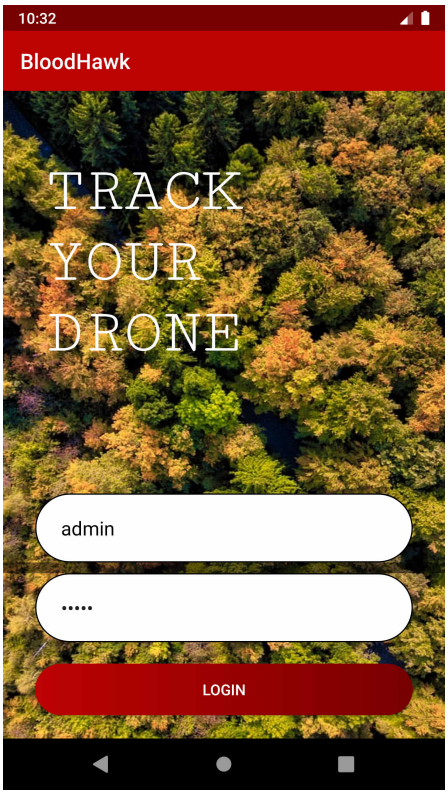
- I applikationen kunna följa drönaren i realtid via karta och videoström.
- I applikationen kunna övervaka rutten.
- I applikationen kunna läsa av batterinivå och uppskattad kvarvarande flygtid.
- I applikationen kunna se relevant logginformation.

Det enda ej uppfyllda kravet är övervakning av rutten. Detta har medvetet uteslutits då det hade kunnat äventyra drönarens säkerhet. För att kunna hämta den planerade

rutten från drönaren måste ett meddelande som begär datan skickas till drönaren. Av säkerhetsskäl stöds enbart envägskommunikation mellan drönaren och applikationen, det vill säga att meddelanden kan inte skickas till drönaren från applikationen för att undvika eventuella kapningar eller dylikt. Det är möjligt att implementera ett meddelandefilter som tillåter att just denna typ av meddelande skickas till drönaren, men inga andra. På grund av tidsbrist blev detta dock inte en del av projektet.

4.2.1 Funktionalitet

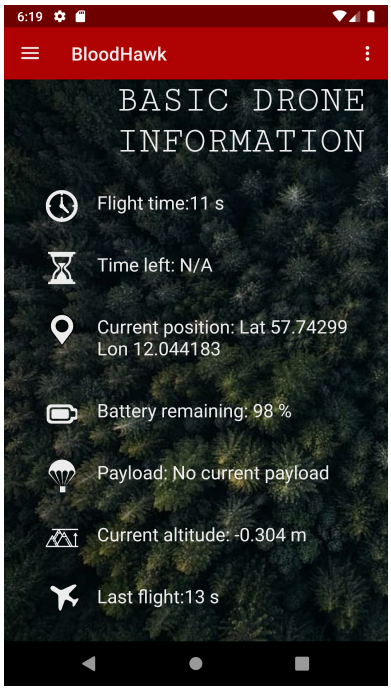
Applikation hämtar data från drönare både i realtid och i form av loggar. När applikation startar syns ett inloggningsfönster, men inloggningssystemet har i skrivande stund endast grundläggande funktionalitet (figur 4.2). Det skapades för att understryka vikten av ett säkert system när man kan komma att behandla känsliga transporter.



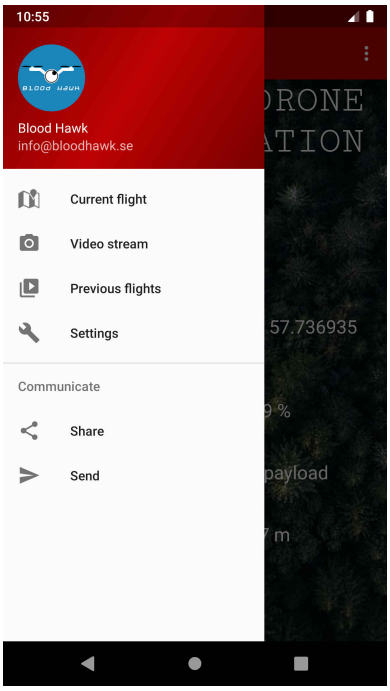
Figur 4.2: Grundläggande inloggningsfönster.

Efter inloggning kommer användaren till en startskärm där grundläggande information finns tillgänglig, såsom flygtid, batterinivå och dylikt (figur 4.3). Denna data uppdateras

automatiskt med information i realtid från drönaren. Från detta fönster kan användaren även navigera till andra lägen som exempelvis kart- eller videoläge (figur 4.4).

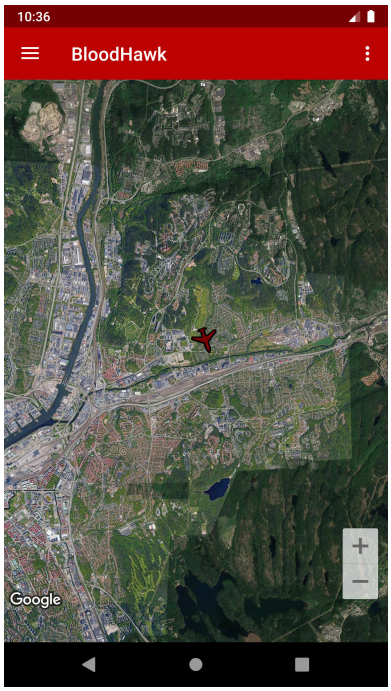


Figur 4.3: Grundläggande data till användaren.



Figur 4.4: En meny flik där man kan växla mellan olika vyer.

För att en användare enkelt skulle kunna följa drönarens position skapades en vy för detta med hjälp av Google Maps. Denna vy tar drönarens nuvarande position och markerar positionen med ett flygplan. Flygplanet roterar för att visa vilken kurs drönaren har (figur 4.5). Om användaren hellre vill övervaka drönaren via videoström kan användaren byta till videovyn som får en ström, med relativt låg fördröjning, på vad drönaren ser (figur 4.6). Det finns även möjlighet till att kombinera bägge vyer och få en tydligare överblick (figur 4.7).



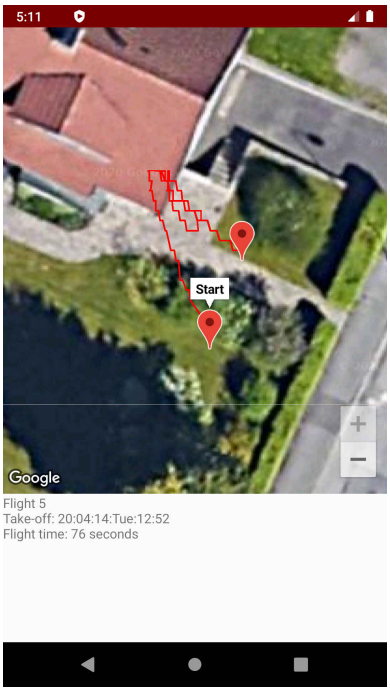
Figur 4.5: En kartvy med drönarens position och riktning utmärkt.

Figur 4.6: En videovy där användaren kan se vad drönaren ser.

Figur 4.7: En kombinerad kartvy och videovy.

Efter en flygning kan användaren vilja kontrollera rutten och loggarna för drönaren. Därför skapas en flyglogg efter varje flygning som användaren kan besöka. Dessa loggar visas först upp i en lista (figur 4.8) och när användaren klickar på någon av loggarna dyker en mer detaljerad vy upp med rutt och annan information (figur 4.9).

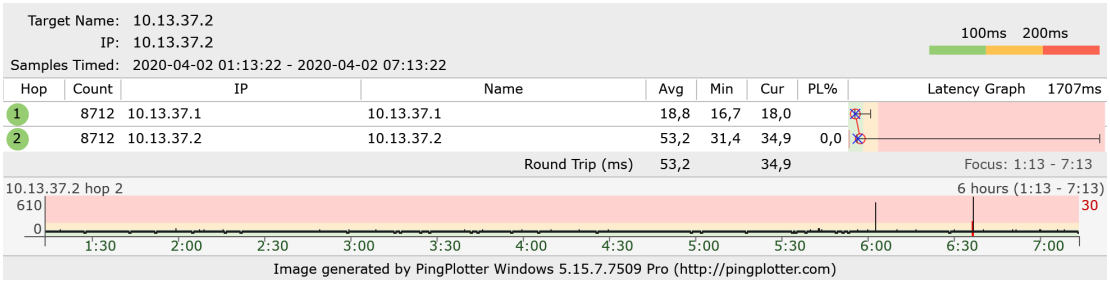
Figur 4.8: En lista med sparade flygloggar.



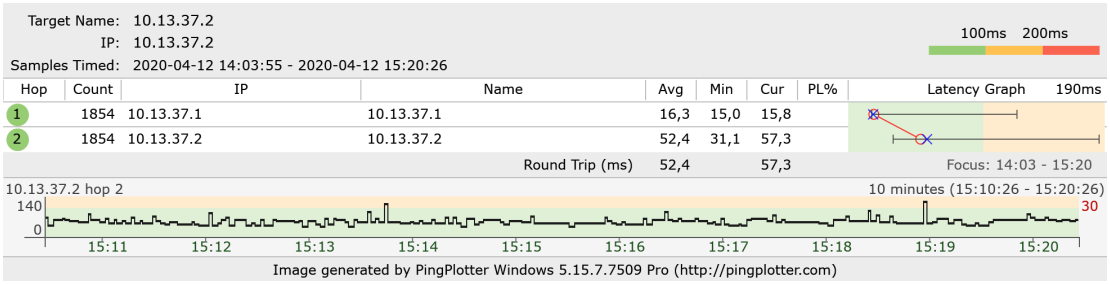
Figur 4.9: En detaljerad flyglogg med bland annat rutt och flygtid.

4.2.2 Backend

Den bakomliggande infrastrukturen fungerade som förväntat. Flera enheter kan samtidigt både se videoströmmen och få ut sensordata från drönaren. Detta samtidigt som fördröjningen till drönaren från markstation är relativt låga.



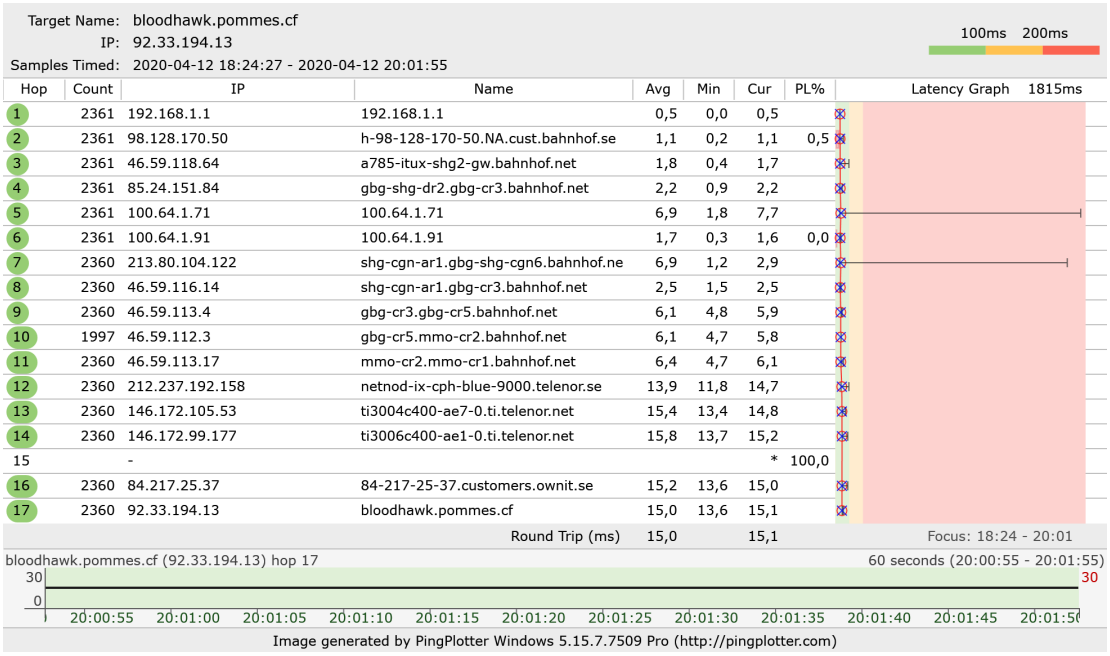
Figur 4.10: PingPlotter-mätning utförd mot drönaren från en bärbar dator ansluten via wifi.



Figur 4.11: PingPlotter-mätning utförd mot drönaren från en kabelansluten stationär dator.

För att få exakta siffror på fördröjningen via 4G och VPN utfördes flera PingPlotter-mätningar mot drönaren från både bärbar och stationär dator. PingPlotter är ett program för att mäta svarstider och paketförluster mot en nätverksenhet över en längre tid [41]. Den visualiserar också mätningarna för enklare analys. Den bärbara datorn var ansluten med wifi och den stationära var kabelansluten. Resultatet blev det förväntade med lite längre svarstider för den bärbara datorn (figur 4.10), med en genomsnittlig svartid på $18{,}8\,\mathrm{ms}$ till servern och $53{,}1\,\mathrm{ms}$ till drönaren. Detta kan jämföras med den stationära datorns $16{,}3\,\mathrm{ms}$ och $52{,}4\,\mathrm{ms}$ till samma mål (figur 4.11). I båda fallen är det som förväntat 4G-länken som är den långsammaste i kedjan. Det förekommer även vissa enskilda tester med väldigt höga svarstider över 4G-länken vilket är icke-önskvärt men ofrånkomligt.

För att få en uppskattning av den extra fördröjningen som uppstår på grund av VPN utfördes en ytterligare mätning (figur 4.12), denna gång mot serverns externa IP, som därefter jämfördes med tidigare mätning. Båda mätningarna utfördes mot samma fysiska server, men via olika nätverksgränssnitt, vilket ger ett överskådligt resultat. Via VPN var svarstiden som tidigare nämnt $16{,}3\,\mathrm{ms}$ i genomsnitt. Detta kan jämföras med $15{,}0\,\mathrm{ms}$ mot serverns publika IP. Slutsatsen blir således att VPN-tunneln introducerar en fördröjning på lite drygt $1\,\mathrm{ms}$ vilket anses vara acceptabelt.



Figur 4.12: PingPlotter-mätning utförd mot serverns externa IP från en kabelansluten stationär dator,

5 Slutsats

Syftet med projektet var att ta fram en drönare kapabel att genomföra autonoma leveransuppdrag mellan Östra sjukhuset och Sahlgrenska sjukhuset, starta och landa vertikalt, samt att utveckla en tillhörande Android-applikation för övervakning av drönaren.

Projektet har varit präglat av vissa genomgående problem. Eftersom projektet initialt bestod av en överväldigande del hårdvarurelaterade arbetsuppgifter blev det mycket väntan under hårdvarans leveranstid. Detta förvärrades än mer av den defekta flygdatorn. Hårdvara beställdes i tron om att flygdatorn fungerade och därför kunde betydande framsteg inte nås förrän efter felsökningen och den efterföljande nybeställningen. För att effektivisera kunde mer bakomliggande arbete ha utförts under denna tid men eftersom applikationen inte var en del av projektet vid den tidpunkten fanns det begränsat med arbete att utföra.

Under projektets gång utbröt pandemin COVID-19 [42] vilket på många sätt försvårade utförandet. Då många av projektets aspekter är hårdvaruberoende fick en plan tas fram

med ett nytt tillvägagångssätt och nya mål som skulle utgöra grunden för fortsättningen av projektet.

I ett tidigt skede beslöts det att all hårdvara skulle flyttas från Chalmers lokaler till en privat plats så att en total nedstängning av universitetet inte skulle förhindra hårdvaran från att nås och användas. Detta resulterade i att allt fysiskt byggande designerades till en gruppmedlem. Resterande medlemmar blev begränsade till att enbart kunna arbeta med mjukvaruaspekter. Då projektet till en början inte innehöll stora mängder mjukvara beslutades det att utöka arbetet med en Android-applikation som skulle ge användaren möjlighet att följa drönaren i realtid samt erhålla data från flygningar.

Kombinationen av de hårdvarurelaterade dröjsmålen, COVID-19 och dess konsekvenser för projektet samt upptäckten av drönarens svaghet mot vind i start och landning innebar att mängden provflygningar blev betydligt färre än önskvärt. Med bättre förhandsstudier om vindpåverkan och val av drönarmodell hade projektet kunnat genomföras smidigare.

Under projektet begicks vissa tydliga misstag som delvis kan förklaras av projektgruppens relativa oerfarenhet när det kommer till ämnet. Bland annat fanns ett återkommande problem med skruvar som lossnar, vilket inte togs i åtanke då liknande projekt inte genomförts tidigare. Här kunde ytterligare litteraturstudier innan montering lett till att låsande eller limmade skruvar använts från början.

I ett försök att minimera budgeten för projektet beställdes enbart ett batteri till drönaren. Då vädret minimerade antalet flygtillfällen hade markant fler testflygningar kunnat utföras om mer än ett batteri funnits tillgängligt.

Den inbyggda kompassen i flygdatorn behövde kalibreras om flertalet gånger och störtes lätt ut av magnetfälten från omkringliggande elektronik. En externt monterad kompass hade motverkat detta, minimerat störningarna och förbättrat drönarens prestanda.

Även om projektet inte nådde upp till alla uppsatta mål finns det fortfarande potential i produkten. Om drönaren skulle kommersialiseras och sättas i produktion bör vissa åtgärder vidtas. Vind och dåligt väder orsakar stora problem för drönaren och det stora vingspannet som möjliggör en relativt energieffektiv framåtdrift utgör en nackdel vid start och landning. Vid reell användning hade det varit fördelaktigt med en drönarmodell

anpassad för VTOL från grunden och inte som en efterkonstruktion, alternativt starta drönaren som ett flygplan utan VTOL-funktion. Drönaren behöver också ha större motståndskraft mot väta och vind, speciellt om den ska arbeta i ett klimat liknande det i Göteborg men med rätt modell finns definitivt potentiella användningsområden. För vårdleveranser skulle driftsäkerheten behöva vara betydligt högre än vad som åstadkoms i projektet.

5.1 Drönare

Drönaren kan styras och övervakas över 4G och fungerar väl som quadcopter. På grund av yttre omständigheter såsom coronaviruset, dåligt väder och fördröjd leverans av motorer har drönarens flygplansläge inte kunnat testas önskvärt och relaterade kravspecifikationer uppfylls dessvärre ej. Drönaren lägger dock en god grund för fortsatt arbete inom autonoma leveransuppdrag.

Dessvärre är videoströmmens fördröjning för hög för att kunna styra drönaren över 4G-nätet. Majoriteten av fördröjningen orsakar själva kameran och är väldigt svårt att påverka. Med facit i hand hade en kamera anpassad för drönare och med lägre fördröjning varit ett bättre val.

5.2 Applikation

I applikationen kan användaren i realtid följa drönaren via både karta och videoström, samt kontrollera batterinivå och flygtid. Efter varje avslutad flygning skapas en flyglogg som möjliggör kontroll av rutten och dylikt. Applikationen uppfyller vår kravspecifikation med undantaget för övervakning av rutt vilket utelämnats av säkerhetsskäl då detta kräver tvåvägskommunikation med drönaren.

5.3 Vidareutveckling

Det finns flera möjligheter till vidareutveckling i projektet. Applikationen kan med fördel utvecklas till fler operativsystem. Det är möjligt att implementera autentisering och kryptering av kommunikationen mellan drönare och applikationen. Med det skulle olika nivåer av användarkonton skapas där möjlighet finns att skapa uppdrag och starta dessa via applikationen. Vidare är det möjligt att koppla drönare till specifika användare. Allt för att möjligöra övervakning av multipla drönare från ett konto, eller att flera

konton övervakar samma drönare.

Drönaren kan även utrustas med en släppfunktion för autonom avlämning av last. Detta skulle med fördel kunna integreras i den uppdaterade modell som diskuteras i avsnitt 5. Detta skulle kunna genomföras antingen genom att släppa från luften med någon fallskärmsanordning eller genom att sväva nära marken, alternativt landa helt för att sedan lyfta och återvända till startpunkten.

I en framtida iterationen kan 5G användas istället för 4G för ytterligare bandbredd, men framförallt för att sänka fördröjningen. Fördröjning är det största praktiska hindret mot att flyga drönaren manuellt via mobilnätverket. Kombinerat med en bättre kamera skulle detta förbättra möjligheterna avsevärt.

Referenser

[1] Wikipedia, ”Unmanned aerial vehicle”, 2020. [Online]. Tillgänglig: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle (hämtad 2020-02-12).

[2] Swiss Post, EOC hospital group, Matternet, ”Further drone flights in Lugano”, 2017. [Online]. Tillgänglig: https://www.post.ch/en/about-us/media/press-releases/2017/further-drone-flights-in-lugano (hämtad 2020-02-11).

[3] Google, ”Google Maps”, 2020. [Online]. Tillgänglig: https://www.google.se/maps (hämtad 2020-05-14).

[4] Innovationsplattformen VGR, ”Framtidens vårdtransporter i luften?”, 2019. [Online]. Tillgänglig: https://www.vgregion.se/siteassets/ovriga_webbplatser/innovationsplattformen/framtidens-vard-transporter-i-luften_slutrapport.pdf (hämtad 2020-02-07).

[5] Ardupilot, ”QuadPlane Overview”. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/quadplane-overview.html (hämtad 2020-04-28).

[6] R. Luppicini och A. So, ”A technoethical review of commercial drone use in the context of governance, ethics, and privacy”, *Technology in Society*, vol. 46, ss. 109–119, aug. 2016.

[7] Polisen, ”Drönare”, 2019. [Online]. Tillgänglig: https://polisen.se/lagar-och-regler/trafik-och-fordon/dronare/ (hämtad 2020-04-28).

[8] Sveriges Riksdag, ”Kameraövervakningslagen och möjligheterna att använda drönare”, 2017. [Online]. Tillgänglig: https://www.riksdagen.se/sv/dokument-lagar/arende/betankande/kameraovervakningslagen-och-mojligheterna-att_H401JuU31 (hämtad 2020-02-06).

[9] Regeringen, ”Så ska samhället fortsätta fungera – även om skolorna skulle behöva stängas”. [Online]. Tillgänglig: https://www.regeringen.se/pressmeddelanden/2020/03/sa-ska-samhallet-fortsatta-fungera--aven-om-skolorna-skulle-behova-stangas/ (hämtad 2020-05-11).

[10] Luftfartsverket, ”Drönarflygning”, 2019. [Online]. Tillgänglig: http://www.lfv.se/tjanster/informationstjanster/lfvs-dronarkarta (hämtad 2020-04-28).

[11] Emlid, ”Hardware setup?”, 2020. [Online]. Tillgänglig: https://docs.emlid.com/navio2/ardupilot/hardware-setup/ (hämtad 2020-04-29).

[12] AC Plastics INC, ”Guide to Plastic Weights”. [Online]. Tillgänglig: https://www.acplasticsinc.com/informationcenter/r/guide-to-plastic-weights (hämtad 2020-04-28).

[13] A. Happe, ”LTE uplink for Raspberry Pi: Huawei E3372 vs Waveshare SIM7600E-H”, 2019. [Online]. Tillgänglig: https://snikt.net/blog/2019/07/05/lte-uplink-for-raspberry-pi-huawei-e3372-vs-waveshare-sim7600e-h/ (hämtad 2020-03-04).

[14] WireGuard, ”WireGuard performance”, 2019. [Online]. Tillgänglig: https://www.wireguard.com/performance/ (hämtad 2020-04-05).

[15] ArduPilot, ”MAVLink Interface”, 2019. [Online]. Tillgänglig: https://ardupilot.org/dev/docs/mavlink-commands.html (hämtad 2020-04-19).

[16] S. A. Alkadhim, ”Communicating with Raspberry Pi via MAVLink”, juli 2017. DOI: 10.13140/RG.2.2.21398.65606.

[17] *Pixhawk 4 GPS Quick Start Guide*, HOLYBRO(H.K.) LIMITED, aug. 2018.

[18] *Pixhawk 4 Pinout*, HOLYBRO(H.K.) LIMITED, aug. 2018.

[19] Emlid, ”Navio2 - autopilot HAT for Raspberry Pi”, 2020. [Online]. Tillgänglig: https://emlid.com/navio/ (hämtad 2020-05-25).

[20] ArduPilot, ”ArduPilot”, 2020. [Online]. Tillgänglig: https://ardupilot.org/ (hämtad 2020-05-25).

[21] ArduPilot, ”PPM Encoder”, 2019. [Online]. Tillgänglig: https://ardupilot.org/copter/docs/common-ppm-encoder.html (hämtad 2020-04-29).

[22] Arduino, ”Arduino Pro Mini”, 2020. [Online]. Tillgänglig: https://store.arduino.cc/arduino-pro-mini (hämtad 2020-04-29).

[23] ArduPilot, ”Loiter Mode”, 2019. [Online]. Tillgänglig: https://ardupilot.org/copter/docs/loiter-mode.html (hämtad 2020-04-29).

[24] ArduPilot, ”Advanced Compass Setup”, 2019. [Online]. Tillgänglig: https://ardupilot.org/copter/docs/common-compass-setup-advanced.html (hämtad 2020-04-29).

[25] ArduPilot, ”ESC Calibration”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/guide-esc-calibration.html (hämtad 2020-04-29).

[26] ArduPilot, ”QSTABILIZE Mode”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/qstabilize-mode.html (hämtad 2020-05-13).

[27] ArduPilot, ”QHOVER Mode”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/qhover-mode.html (hämtad 2020-05-13).

[28] ArduPilot, ”QLOITER Mode”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/qloiter-mode.html (hämtad 2020-05-13).

[29] ArduPilot, ”FBWA Mode (FLY BY WIRE_A)”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/fbwa-mode.html (hämtad 2020-05-13).

[30] ArduPilot, ”Flying a QuadPlane”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/quadplane-flying.html (hämtad 2020-05-13).

[31] ArduPilot, ”LOITER Mode”, 2019. [Online]. Tillgänglig: https://ardupilot.org/plane/docs/loiter-mode.html (hämtad 2020-05-13).

[32] Freedesktop, ”GStreamer RTSP Server”, 2020. [Online]. Tillgänglig: https://gstreamer.freedesktop.org/modules/gst-rtsp-server.html (hämtad 2020-03-04).

[33] dest-unreach.org, ”socat - Multipurpose relay”, 2020. [Online]. Tillgänglig: http://www.dest-unreach.org/socat/ (hämtad 2020-04-29).

[34] Dronecode, ”MAVSDK-Java”, 2020. [Online]. Tillgänglig: https://github.com/mavlink/MAVSDK-Java (hämtad 2020-04-15).

[35] DroneKit, ”DroneKit-Android”, 2020. [Online]. Tillgänglig: https://github.com/dronekit/dronekit-android (hämtad 2020-04-20).

[36] dronefleet, ”dronefleet-java”, 2020. [Online]. Tillgänglig: https://github.com/dronefleet/mavlink (hämtad 2020-04-25).

[37] Dronecode, ”MAVSDK”, 2020. [Online]. Tillgänglig: https://mavsdk.mavlink.io/develop/en/ (hämtad 2020-04-19).

[38] Dronecode, ”FAQ”, 2020. [Online]. Tillgänglig: https://mavsdk.mavlink.io/develop/en/getting_started/faq.html (hämtad 2020-04-05).

[39] ReactiveX, ”RxJava”, 2020. [Online]. Tillgänglig: https://github.com/ReactiveX/RxJava (hämtad 2020-04-19).

[40] Raspberry Pi Forums, ”How to get lowest latency with PiCamera”, 2019. [Online]. Tillgänglig: https://www.raspberrypi.org/forums/viewtopic.php?t=240390 (hämtad 2020-05-07).

[41] Pingman Tools, ”PingPlotter”, 2020. [Online]. Tillgänglig: https://www.pingplotter.com/ (hämtad 2020-04-29).

[42] WHO, ”WHO Director-General’s opening remarks at the media briefing on COVID-19 - 11 March 2020”, 2020. [Online]. Tillgänglig: https://www.who.int/dg/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19---11-march-2020 (hämtad 2020-04-28).

Institutionen för data- och informationsteknik
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sverige 2020
www.chalmers.se

CHALMERS