CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Autonom hybriddrönare med personsdetektering för sök- och räddningsuppdrag

Kandidatarbete inom civilingenjörsprogrammet

Kalle Arvidsson, Anton Claesson, Emma Johansson,
Samuel Rörgren, Sebastian Svanland, Sara Tell

Institutionen för Data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2019

Kandidatarbete datx02-19-86

Autonom hybriddrönare med personsdetektering för sök– och räddningsuppdrag

Kalle Arvidsson
Anton Claesson
Emma Johansson
Samuel Rörgren
Sebastian Svanland
Sara Tell

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2019

Autonom hybriddrönare med personsdetektering för sök- och räddningsuppdrag

Handledare: Roger Johansson, Institutionen för Data- och informationsteknik
Examinator: Sven Knutsson, Institutionen för Data- och informationsteknik

Kandidatarbete DATX02-19-86
Institutionen för Data- och informationsteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Telephone +46 31 772 1000

Typeset in LaTeX
Göteborg, Sverige 2019

Abstract

Lately, UAVs (Unmanned Aerial Vehicles), also known as drones, have become very popular among both hobbyists and professionals alike. This is especially true for rotor wing drones, which possesses the ability to hover and to start and land vertically. Fixed wing drones are not as easily maneuvered, but instead possesses the ability to glide through the air using their wings, which is a more energy efficient way of flying. The aim of the work presented in this report was to develop a hybrid drone, equipped with a human detection system and capable of autonomous flight in order to be used in search- and rescue operations where both longer flight time and high maneuverability can be of use. The hybrid was constructed by mounting four vertically directed rotors upon a fixed wing drone. The final prototype is able to fly and follow a simple route using a Pixhawk 4 flight controller. A real-time video capturing and wireless transmission software was developed in parallell with an object detection system utilizing deep machine learning algorithms. The real-time video system was never fully finished, but the object detection system can detect humans with varying results depending on situation of use, and can with relative ease be upgraded for better results. In order to reach a further level of autonomous flight, RRT-algorithms (Rapidly exploring Random Tree) were examined for path planning. The work resulted in a path planning algorithm where static, known obstacles can be avoided. Tests and simulations were made for the algorithm, but it was never fully implemented on the drone. Although progress has been made during this project, subsequent development is required if the goal of the project is to be obtained.

Sammandrag

Obemannade flygfarkoster har på senare tid blivit ett mycket populärt fenomen, både bland hobbyister och för professionellt bruk. Detta gäller främst rotordrönare, som har möjligheten att både hovra och utföra vertikal start- och landning. Flygplansdrönare är inte lika lättmanövrerade, men besitter istället förmågan att kunna glidflyga vilket är ett mer energieffektivt sätt att flyga på. Arbetet som redovisas i denna rapport syftar till att utveckla en autonom hybriddrönare, utrustad med ett människodetekteringssystem. Detta för att kunna användas i sök- och räddningsuppdrag där både lång flygtid och hög manövrerbarhet kan vara till nytta. Hybriden konstruerades genom att montera fyra vertikalt riktade rotorer på en flygplansdrönare. Den slutgiltiga prototypen kan flyga och följa en enkel rutt genom användandet av en Pixhawk 4 flygkontroller. En trådlös realtidsvideo- och överföringsmjukvara utvecklades parallellt med ett objektsidentifieringssystem som använder algoritmer för djup maskininlärning. Realtids-videosystemet blev aldrig färdigställt, men objektsidentifieringssystemet kan med varierande resultat upptäcka människor beroende på användningssituation, och kan på ett relativt enkelt sätt uppgraderas för bättre resultat vid vidareutveckling. RRT-algoritmer (Rapidly exploring Random Tree) undersöktes för ruttplanering i syfte att nå en högre nivå av autonom flygning. Arbetet resulterade i en ruttplaneringsalgoritm för undvikning av fasta kända hinder. Tester och simuleringar gjordes för algoritmen, men den implementerades aldrig fullt ut på drönaren. Även om det slutgiltiga resultatet har kommit en bit på vägen till det önskade slutmålet kvarstår mycket vidareutveckling innan systemet kan färdigställas.

Förord

Nedan nämnda personer och grupper har varit till stor hjälp under projektets gång och vi vill därför ge dem ett stort tack.

Vi vill tacka vår handledare Roger Johansson för god kommunikation och rådgivning genom hela arbetet. Vidare vill vi tacka Lars Norén för all hjälp med inköp av komponenter och material. Vi vill även tacka Fackspråk för deras vägledning och hjälp med skrivandet av denna rapport samt David Frisk för tillhandahållande av LaTeX-mall. Slutligen vill vi tacka kanditatarbetsgruppen DATX02-19-13 för en väl fungerade disponering av det gemensamma projektrummet. Det har varit lärorikt och roligt att ta del av varandras projekt och utbyta erfarenheter.

Notera: alla figurer och tabeller i rapporten är skapade av författarna själva om inget annat anges.

Göteborg, Maj 2019

Nomenklatur

Begrepp	**Förklaring SVE**
Darkflow	Open-source ramverk för användning av YOLO tillsammans med Tensorflow
Darknet	Open-source ramverk för YOLO kompilerat i C och CUDA
Github	Webbhotell för programvaruutvecklingsprojekt
LabelImage	Grafiskt verktyg för manuell generering av annoteringsdata som kan användas i maskininlärningsapplikationer
Maskininlärning	Metod för att få datorer att *lära sig* baserat på data utan att datorerna har programmerats för den specifika uppgiften
On-board computer	Datorn ombord på objektet, i detta fall ombord/-fastmonterad på drönaren
Open-source	Öppen källkod: datorprogram vars källkod är tillgänglig att användas, läsas, modifieras och vidaredistribuera för den som vill.
Port forwarding	Ett sätt att vidarebefodra särskilda inkommande anslutningar till särskild enhet
Python/C	Programmeringsspråk
Tensorflow	Open-source maskininlärningsbibliotek

Förkortning	**Engelska**	**Förklaring SVE**
ASM	*Auxiliary Signal Module*	Signalmodulen
CNNs	*Convolutional Neural Networks*	En typ av neuralt nätverk som ofta används vid bildbehandlingstillämpningar.
Core ML	*Core Machine Learning*	Ett maskininlärningsramverk utvecklat av Apple Inc.
CPU	*Central Processing Unit*	Processor
CUDA	*Compute Unified Device Architecture*	NVIDIA's arkitektur för parallellbearbetning av data i sina GPU-enheter
DNS	*Domain Name System*	En uppslagsenhet som översätter ett domännamn till en faktisk IP-adress.
ESC	*Electronics Speed Control*	Elektroniskt fartreglage
FWD	*Fixed Wing Drone*	Drönare i form av ett flygplan
GPU	*Graphics Processing Unit*	Mikroprocessor som sitter på en dators grafikkort
HTTP	*Hyper Text Transfer Protocol*	Ett protokoll för att överföra data.
IP-adress	*Internet Protocol address*	Internetadress
OpenCV	*Open Source Computer Vision Library*	Programvarubibliotek med rutiner inom datorseende och bildanalys
PMM	*Power Management Module*	Krafthateringsmodul
PPM	*Pulse Position Modulation*	Signalkonverterare
RC	*Radio Controll*	Radio kontroll
RRT	*Rapidly exploring random tree*	En algoritm för att hitta en väg mellan två punkter och samtidigt kunna undvika kända hinder.
RTSP	*Real Time Streaming Protocol*	Ett protokoll skapat för att strömma media över internet.
RWD	*Rotor Wing Drone*	Rotordrönare
TCP	*Transmission Control Protocol*	Ett protokoll i hur data överförs på ett förlustfritt sätt.
UAV	*Unmanned Arial Vehicle*	Obemannad luftfarkost
UDP	*User Datagram Protocol*	Ett protokoll i hur data överförs på ett snabbt och förlustriskt sätt.
URL	*Uniform Resource Locator*	En textsträng som identifierar en nätverksresurs inom ett lokalt nätverk eller på internet.
VTOL	*Vertical Take-Off and Landing*	Vertikal start och landning
YOLO	*You Only Look Once*	Ett open-source objektsidentifierings- och klassificieringssystem

Innehåll

Figurer

1
Inledning

Obemannade luftfarkoster, även kallade UAV:er (Unmanned Aerial Vehicles) eller drönare, är ett samlingsnamn på motorförsedda luftfartyg som kan flygas autonomt eller fjärrstyras [1]. Dessa kan delas in i två subkategorier. Den ena drönartypen har fixerade vingar likt ett flygplan och kallas Fixed Wing Drone (FWD). Den andra sorten har propellrar, som kan skilja i antal, och liknar mer en helikopter. Denna typen kallas för Rotor Wing Drone (RWD) eller rotordrönare.

Historiskt sett har utvecklingen av system och plattformar för UAV:er främst skett i militärt syfte [2]. I dagens läge är dock drönare populära inom en mängd olika applikationsområden. Kommersiellt sett är RWD mycket intressanta för övervaknings-, fotografi- och transportbranschen [3]. Ytterligare exempel på ett användningsområde där tekniken kan tillämpas är inom jordbruket, där specialiserade kameror och molnbaserade analysverktyg möjliggör för jordbrukare att kontinuerligt övervaka sina skördar. Liknande plattformar gör det även möjligt för byggföretag att övervaka utvecklingen av sina byggprojekt i realtid, eller för gruvföretag att erhålla precisa volymdata av utgrävningsplatser [4]. På grund av den kompakta designen, en hög nivå av manövrerbarhet samt svävningsegenskapen hos RWD, som tillåter en stabil och precis flygning, är dessa drönare speciellt användbara för denna typ av tillämpningar [3].

Det finns många fler teoretiska tillämpningsområden av drönarteknologin, men dagens kommersiellt tillgängliga drönare har inte en tillräcklig nivå av autonom kontroll för att utföra några av de beskrivna uppdragen utan övervakning från en skicklig mänsklig operatör, vilket i många fall gör användandet av drönare till en långsam, dyr och farlig process [4].

En annan begränsning för alla typer av rörliga robotsystem är den begränsade mängd energiresurser som kan bäras med vid förflyttning. Detta problem blir än mer uppenbart bland drönare, då flygning i sig kräver en relativt låg vikt i förhållande till farkostens lyftkraft. Batteriets storlek och kapacitet behöver därför kompromissas vilket minskar flygtiden. Utöver detta är även vissa flygmönster och manövrar kostsamma energimässigt utifrån ett reglerperspektiv. Att få en RWD att sväva är exempelvis ett problem som kräver en relativt stor del av den tillgängliga energin för beräkning och styrning av kontrollsystemet [4]. Detta leder till ett av de mest grundläggande problemen för RWDs; förmågan att upprätthålla flygning under en längre period. Denna begränsas av en hög energiförbrukning i kombination med en låg energitillgång, vilket minskar möjligheterna att utföra mer högautomatiserade

uppdrag.

Även om FWD inte besitter samma önskvärda manövrerbarhetsegenskaper som RWD är just energikonsumtionen en viktig anledning till varför de hellre används inom vissa applikationsområden. FWD har en mycket effektivare energiförbrukning än RWD då den flygplansliknande utformningen tillåter dem att generera lyftkraft och glidflyga genom luften. Detta gör dem lämpliga att användas vid kartläggningsuppdrag, eller andra uppgifter där stora ytor behöver täckas.

En kombination av dessa typer av drönare hade eventuellt kunnat erhålla möjligheten till att både starta och landa vertikalt (VTOL, eng. Vertical Take Off and Landing), hålla sig svävande i luften och samtidigt kunna flyga långa sträckor. Det skulle därför finnas en stor potential för vidareutvecklingar vid en fungerande konstruktion av en sådan hybriddrönare.

1.1 Syfte

Detta arbete syftar till att undersöka, dokumentera och utveckla ett högautomatiserat system för användning i sök- och räddningsuppdrag. Systemet skall grunda sig i konstruktion av en hybrid mellan en rotordrönare och en flygplansdrönare med ett objektsidentifieringssystem. Detta görs med avsikt att erhålla en autonom hybriddrönare som kombinerar de bästa egenskaperna från både rotor- och flygplansdrönare samt ha möjligheten att identifiera objekt.

1.2 Problembeskrivning

Ett automatiserat sök– och räddningssystem implementerat på en drönare kräver flertalet olika sammanlänkade delar för att fungera, utöver konstruktion av en flygande drönarkropp. Drönaren bör kunna framföras autonomt. Vidare behöver systemet även kunna detektera människor på egen hand. Därför skall också en videoström i realtid tagen ifrån drönarens perspektiv implementeras, vilken kan visas för användaren där detekterade personer är utmärkta. På grund av detta har projektet delats upp i fyra olika områden som behandlas.

1.2.1 Konstruktion och konfiguration av hybriddrönare

Detta delproblem grundar sig i framtagningen av ett koncept som syftar i att vidare kunna konstruera en flygfärdig prototyp, vilken kan kombinera VTOL- och hovringsegenskapen hos RWD med glidflygningsmöjligheterna hos FWD. I detta innefattas även sammankoppling av nödvändiga elektroniska komponenter, konfiguration av dessa samt implementation av ett styrsystem.

1.2.2 Trådlös kommunikation med drönaren

För att en operatör skall kunna skicka navigationskommandon till drönaren och dessutom ta emot information om upptäckta människor, krävs det att kommunikation mellan drönaren och operatören upprättas. Det krävs också implementation av ett mjuk- och hårdvarusystem med möjlighet att spela in video från drönarens perspektiv. Videodatan som spelas in måste kunna skickas till operatören, före eller efter applicering av objektsidentifiering, för att dels underrätta om upptäckta personer men också för att ge information om pågående flygning som kan vara till nytta.

1.2.3 Autonom flygning

Den färdiga hybriddrönaren skall kunna förflytta sig autonomt mellan olika vägpunkter som väljs av en operatör. Detta innebär först och främst att autonom navigering behöver implementeras. Utöver detta bör drönaren autonomt hantera start och landning. Det är även önskvärt att drönaren har möjlighet att väja inför hinder i flygvägen.

1.2.4 Objektsidentifiering för detektering av människor

Under detta delproblem undersöks möjligheten till att använda maskininlärning för att upptäcka personer i den erhållna videoströmmen. En objektsidentifieringsmodell anpassad för att detektera människor behöver därför utvecklas. Vid utveckling av detta delsystem måste hänsyn tagas till att den fysiska ytan och volymen i drönarkroppen är begränsad, eftersom denna delas med samtliga elektronikkomponenter samtidigt som drönarens totalvikt bör minimeras för att inte kompromissa för mycket med flygtid och flygfärdighet.

1.3 Avgränsningar

Projektet avgränsar sig från att utveckla ett reglersystem för hybriddrönaren, då system för rotordrönare är mycket komplexa och alltför tidskrävande att utveckla från grunden inom ramen av detta projekt. I stället implementeras reglersystemet med färdig fast programvara, vilken kan laddas upp på en flygkontroller.

Ett mål med hybriddrönaren som utvecklas i detta projekt är att den främst skall färdas i flygplansläge för att spara energi. Rotorläget skall utnyttjas vid start och landning samt vid behov av att sväva i luften. Kollisionsundvikning vid autonom flygning är därmed mer relevant i flygplansläge, vilket är vad som kommer undersökas i detta arbete.

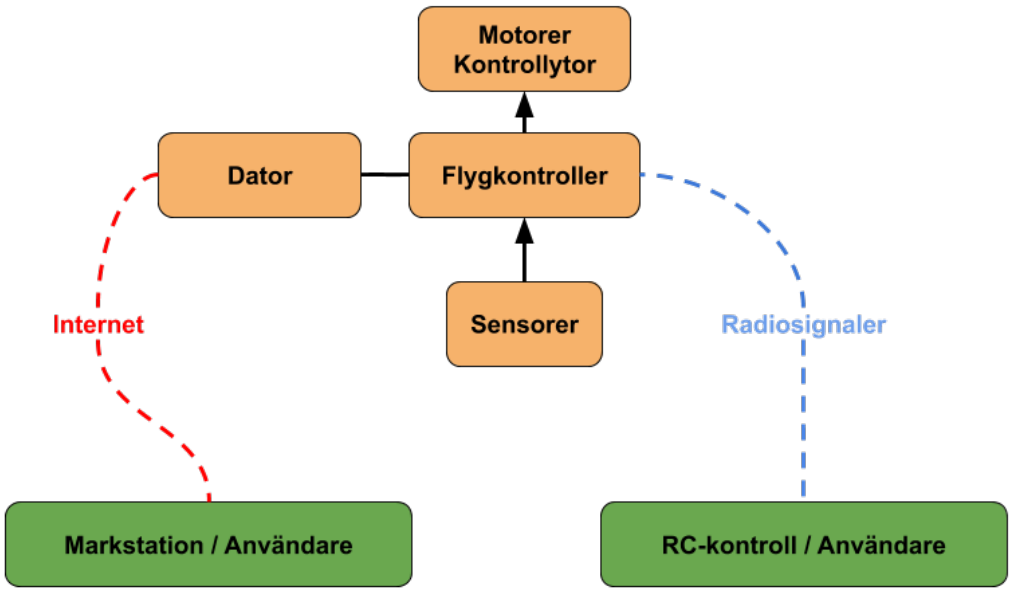
Objektsidentifiering med hjälp av maskininlärning är en väldigt krävande process rent beräkningsmässigt, och därav förutsätts det under projektets gång att de datorer som skall användas för detta i en slutprodukt har tillräcklig beräkningskapacitet

[5]. Av samma skäl avgränsar projektet sig ifrån att implementera objektsidentifieringsberäkningarna direkt ombord på drönaren innan videoströmmen skickas till operatören. I stället utvecklas systemet för att kunna utföra dessa beräkningar vid en markstation, vilken bör bestå av en dator som möter kraven med avseende på hårdvara.

Vidare är ett vanligt tillvägagångssätt för sök- och räddningsuppdrag att använda sig utav värmekamera [6]. Projektet syftar istället till att undersöka möjligheterna för en vanlig färgvideoström. I kapitel 8.4 i diskussionen diskuteras detta mer utförligt.

1.4 Övergripande system

Det inledande arbetet resulterade i en vision om ett övergripande system att förhålla sig till vid fortsatt utveckling. Systemet kan ses i figur 1.1 nedan. För att kunna reglera drönaren och därmed hålla den stabil i luften krävs en flygkontroller som skickar styrsignaler till drönarens motorer och kontrollytor. Denna behöver sensorer som bidrar med mätdata som krävs för att utföra nödvändiga beräkningar. Det behövs även en RC-kontroll för att kunna styra drönaren vid tester och manuell flygning. Detta då lagen kräver att en operatör skall kunna ta över från autonom flygning vid varje givet tillfälle, så länge operatören inte har speciellt tillstånd för detta [7]. Vidare krävs ytterligare en dator ombord för att sköta kommunikation med en markstation. Denna markstation skall ta emot videoström och flygdata, applicera objektsidentifieringssystemet, samt skicka information om aktuellt uppdrag som krävs för autonom flygning.



Figur 1.1: Övergripande struktur över det kompletta system som projektet avser utveckla.

1.5 Rapportens struktur

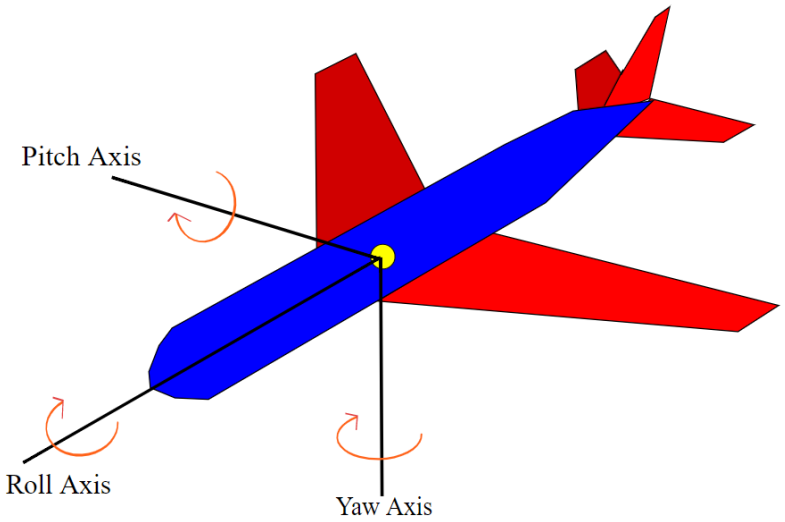
I det här delkapitlet beskrivs rapportens övergripande struktur och hur de olika delarna hänger ihop. Inledningsvis, i kapitel 2, presenteras teori för att ge läsaren grundläggande förkunskaper för att ta del av rapportens innehåll. I kapitel 3-6 beskrivs metod och genomförande för de olika delproblemen. Kapitel 3 behandlar drönarens konstruktion och hårdvara samt beskriver även genomförandet av de flygtester som testar hårdvara. För att kommunicera med drönaren användes en trådlös anslutning över internet, som beskrivs i kapitel 4. Kapitel 5 behandlar en algoritm och programvara för att tillgodose autonom flygning samt flygtester som testar programvaran. Kapitel 6 beskriver hur objektidentifiering för att identifiera människor utvecklades med hjälp av maskininlärningsalgoritmer och hur dessa testas. Genomförandet av testerna från kapitel 6 samt genomförandet av flygtesterna från kapitel 3 och 5 beskrivs i respektive kapitel medan resultaten från dessa presenteras sammanställt i kapitel 7. Rapporten avslutas med diskussion och slutsats i kapitel 8 respektive 9.

2
Teori

I detta kapitel beskrivs teori och fakta som ligger till grund för arbetet och som ämnar ge läsaren den bakgrundskunskap som anses nödvändig för fortsatt läsning. För att förenkla läsningen av de resterande delarna i rapporten inkluderar denna delen grundläggande kunskaper om lyftkraft, manövrering och typer av drönare. Utöver detta beskrivs ruttplaneringsalgortimen RRT som i kapitel 5 ligger till grund för utvecklingen av den egna ruttplaneringsalgoritmen. Bakomliggande teori kring nätverkskommunikation för videoströmning samt teori kring objektidentifiering och maskininlärning presenteras också.

2.1 Lyftkraft och manövrering

En luftfarkost har tre axlar att förhålla sig till: pitch, yaw och roll [8]; vilket figur 2.1 illustrerar. Dessa kan regleras med hjälp av gravitationen och luftmotståndet och därmed bestämma luftfarkostens flygriktning i rummet.



Figur 2.1: Rotationsaxlar på en luftfarkost. Från [9]. CC-BY-SA.

Flygplan och helikoptrar är luftfarkoster som använder luftens dynamiska egenskaper för att producera lyftkraft. Den grundläggande principen som utnyttjas orsakas av två fysikaliska fenomen: fysisk deflektion och Bernoullis princip [10]. Dessa leder

till att fasta föremål som befinner sig i en fluid (vätska eller gas) orsakar omdirigering av fluidens flöde vilket i sig genererar kraft enligt Newtons tredje lag. För att detta ska ske måste föremålet vara i direkt kontakt med fluidens flöde och föremålets relativa hastighet till flödet av fluiden skiljt från noll. Faktorer som påverkar kraftgenerationen kan försummas till densiteten och viscositeten hos fluiden, föremålets form samt den relativa hastigheten mellan föremålet och fluidflödet.

För att maximera lyftkraften som genereras av rörelse använder både flygplan och rotorkraft sig av vingar. Lyftkraften som genereras av en vinge kan beskrivas med följande ekvation

$$L = \frac{C_l \cdot A \cdot \rho \cdot v^2}{2} \tag{2.1}$$

där C_l är vingens lyftkoefficient, A är vingens area, ρ är luftens densitet och v den relativa hastigheten mellan vingen och luftflödet [11]. Lyftkoefficienten av en vinge beror på dess vingprofil och anfallsvinkel [12]. Anfallsvinkeln är vinkeln mellan vingens korda(linjen från vingens främre kant till bakre kant) och luftströmmens riktning. Lyftkoefficienten och därmed lyftkraften på en vinge ökar med anfallsvinkeln tills dess att ett kritiskt värde uppnås [13]. En vanlig lyftkoefficient vid lägre hastigheter är omkring 1,4 [14]. Om anfallsvinkeln skulle överskrida detta värde minskas lyftkoefficienten och därmed lyftkraften drastiskt. Detta fenomen kallas för överstegring. Vidare så måste flygplan hålla en minimal hastighet för att generera tillräckligt med lyftkraft för att hålla sig luftbundet, eftersom dess lyftkraft framförallt produceras av dess egna hastighet. Av samma skäl behöver ett flygplan tillräckligt lång startbana för att accelerera till sin gränshastighet vid start.

2.1.1 Flygplan och FWD

Ett konventionellt flygplan, eller FWD som baseras på samma egenskaper, är designat att alltid färdas framåt med en anfallsvinkel inom den kritiska gränsen för överstegring. Rörelsen framåt skapar ett luftflöde relativt vingarna och därmed utvecklas en lyftkraft. Genom att justera vinkeln för de olika styrytorna som finns på planet kan vridmoment skapas genom de aerodynamiska krafter som utvecklas, vilket innebär att planet går att manövrera [15].

För att ett flygplan ska behålla sin altitud under flygning måste den minst hålla sin kryssninghastighet, den lägsta hastigheten där lyftkraften som genereras av dess vingar neutraliserar dess vikt. För att hålla denna hastighet behöver flygplanet dragkraft som balanserar ut luftmotståndet [16]. Luftmotståndet på ett föremål påverkas av föremålets form och kvadraten av den relativa hastigheten mellan föremålet och luften [17]. Det betyder att dragkraften för att balansera ut luftmotståndet kan minskas genom att anpassa flygplanets form, eller minska kryssningshastigheten genom att minska flygplanets massa och öka dess vingarea.

2.1.2 Helikopter och RWD

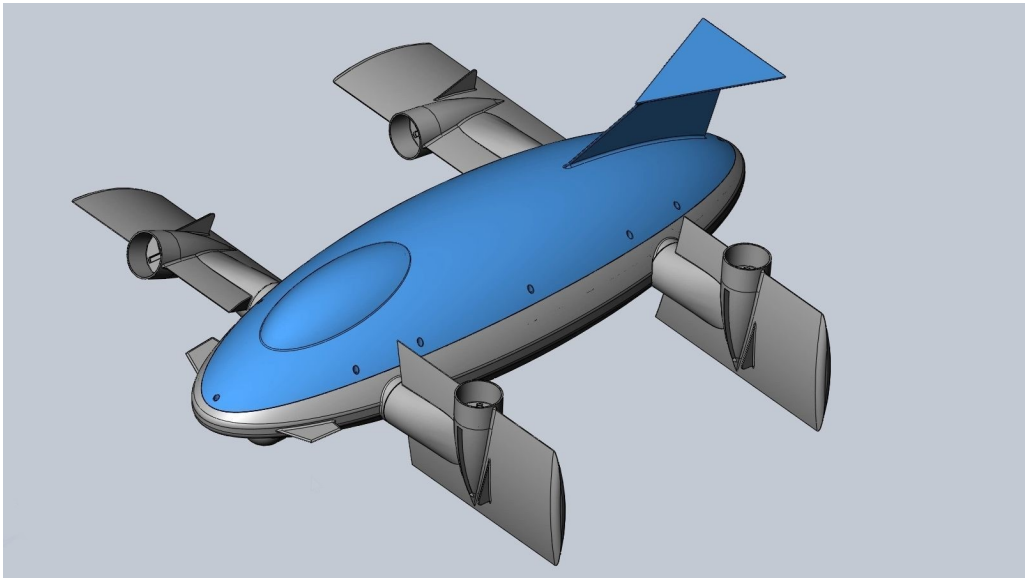
En helikopter, eller RWD som baseras på samma egenskaper, genererar lyft genom att rotera sina rotorblad kring en vertikal axel [18]. Den lyftgenererande rotationen av rotorbladen drivs av motorer. Detta tillåter en helikopter att lyfta även om dess horisontella hastigheten är noll. För helikoptrar med tre eller fler rotorer kan vridmoment för manövrering genereras av differensen av dragkraften som produceras av de olika rotorerna. För helikoptrar med två eller färre rotorer behöver manövrering kompletteras genom att justera rotorbladets anfallsvinkel cykliskt. Detta genererar vridmoment från differensen av lyftkraften vid olika stadier av rotorbladets rotation.

För att minimera den effekt som krävs av motorn till att generera en bestämd mängd dragkraft bör rotorns rotationsarea maximeras [18]. Genom att miniminera massan på farkosten kan dragkraften som krävs för att lyfta farkosten, och därmed motorernas effekt, minskas vidare.

2.1.3 Befintliga hybrider mellan RWD och FWD

Det finns huvudsakligen tre konventionella typer av RWD/FWD hybrider: tiltwing, tailsitter och quadplane. I denna sektion förklaras dessa hybriders funktioner och utformningar, samtidigt som för– och nackdelar för respektive typ diskuteras.

En tiltwing-drönare, enligt figur 2.2, övergår mellan vertikalt och horisontellt läge genom att ändra riktning på sina propellrar relativt till flygkroppen [19]. Denna design ger upphov till extra kontrollfrihet, vilken tillåter den att utföra manövreringar som andra designer inte kan utföra. En konsekvens av detta är minskad robusthet som orsakas av extra rörliga delar.



Figur 2.2: En tiltwing-drönare. Från [20]. Återgiven med tillstånd.

Figur 2.3 visar en tailsitter, en typ av hybrid som baserar sitt flygläge på flygkroppens orientation under flygning [21]. Vid start och landning är flygkroppen riktad uppåt, ståendes på vingarna, med propellrar som får den att lyfta. Väl uppe i luften roterar den 90 grader och övergår i att flyga horisontellt med extra lyftkraft från vingarna. Denna design är robust och tillåter en liten profil i det horisontella planet vid rotorläge och därmed minskar luftmotståndet vid start. Däremot påverkas tailsittern lätt av vindar i sidled som kan störa dess stabilitet.

Figur 2.3: En tailsitter-drönare. Från [22].

Ett quadplane är en hybrid som har separata propellrar avsedda för vertikal flygning och horisontell flygning [23]. Den utgår från en FWD, men med fyra extra propellrar för att lyfta vertikalt. Detta tillåter övergången mellan vertikal och horisontell flygning att ske endast genom att ändra motorernas effekt. Flygningen kan optimeras genom att använda olika typer av propellrar för vertikal respektive horisontell flygning.

Figur 2.4: Ett quadplane. Från [24].

2.2 Nätverksdetaljer

Det finns flera sätt att trådlöst återkoppla en videoström från en flygande drönare till en användare. En möjlig lösning är att använda sig av kommunikation över ett 3G– eller 4G–nätverk. Detta ger möjlighet till att skicka både sensordata och styrsignaler samt videodata mellan drönare och användare.

För att implementera en videoström från en drönare handlar det främst om att överföra bilddata tillräckligt snabbt för att operatören ska kunna se förändringar som sker i lämplig uppdateringsfrekvens. Om målet exempelvis är att användaren manuellt ska kunna undvika hinder som uppkommer i flygvägen, måste uppdateringfrekvensen för videoströmmen vara tillräckligt hög för att användaren skall kunna identifiera hindret genom kameran och sedan agera utefter detta.

På liknande sätt kräver andra tillämpningsområden inte alls samma höga uppdateringsfrekvens. En autonomt flygande drönare som styrs med GPS, vars funktion är att kartlägga en kuststräcka, hade kunnat skicka en ny bild med lägre uppdateringsfrekvens och ändå slutföra uppdraget framgångsrikt.

För att upprätta en videoström från en drönare kan därför flera olika protokoll användas beroende på applikationsområdet. Från OSI-modellen (Open Systems Interconnection model) [25] finns två lager av särskilt intresse i detta syfte: applikationslagret och transportlagret.

I transportlagret finns två särskilda protokoll att ta hänsyn till: User Datagram Protocol UDP[26] och Transmission Control Protocol TCP[27]. UDP är ett protokoll

som är snabbt men som inte hanterar förluster av skickade paket. Detta gör att all data som skickas genom UDP inte är garanterat att komma fram. TCP däremot övervakar skickade paket och skickar paketet på nytt om det inte kom fram. Konsekvensen av detta blir att TCP är betydligt långsammare än UDP.

När det gäller att strömma video i realtid måste bilddata överföras tillräckligt snabbt, annars kommer mottagaren att uppleva avbrott. Då nya bildramar skickas hela tiden ersätter en ny ram snabbt den gamla – på så vis är UDP ett mer motiverat protokoll då paket som gått bort i transport inte längre är relevanta. Har man videoströmmning med en lägre uppdateringsfrekvens kan det bli aktuellt med en TCP-anslutning även om det tar längre tid. Normalt är UDP protokollet som normalt används för strömning just för att UDP är snabbare.

I applikationslagret finns det flera protokoll man kan använda. Två stycken är Hyper Text Transfer Protocol HTTP[28] och Real Time Streaming Protocol RTSP[29]. RTSP är till skillnad från HTTP gjord för att strömma realtidsmedia och ger möjlighet att använda UDP istället för TCP på ett fördelaktigt sätt. Trots att båda protokollen kan användas för att strömma video är HTTP lite sämre utifrån just denna aspekt. Däremot så finns det fördelar med HTTP som RTSP inte har. Till exempel så har HTTP ett brett webbläsarstöd, vilket förenklar möjligheten att implementera en realtidsvideoströmmning där tittare kan se strömmen direkt i en webbläsare utan att särskild mjukvara behöver involveras.

Vid användning av protokollen kan man stöta på anslutningsproblem som involverar port forwarding. Detta fenomen hittar man i nätverk med datorer som är avsedda att ta emot anslutningar och agera som en webbserver. Port forwarding innebär att en nätverksförmedlare ställs in på att skicka vidare särskild trafik till en viss dator i nätverket.

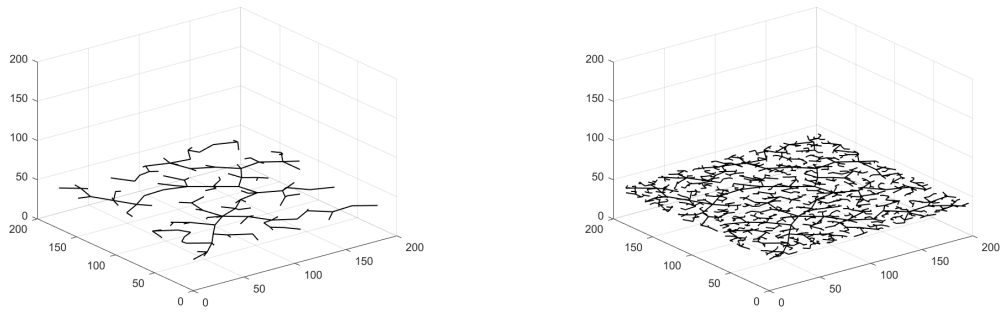
2.3 Ruttplaneringsalgoritmer

Det finns ett flertal olika algoritmer inom robotik för ruttplanering (eng. path planning). Dessa används för att planera hur en robot ska röra sig i en känd miljö. Rapidly Exploring Random Tree (RRT) är en ruttplaneringsalgoritm som bland annat utnyttjas för drönare [30], [31] och som beskrivs mer ingående i 2.3.1 nedan. D* och A* är två andra ruttplaneringsalgoritmer.

Användning av RRT-algoritmen lämpar sig bra för tillfällen då det är önskvärt att erhålla en väg mellan en startpunkt och en slutpunkt relativt snabbt och där denna vägen inte behöver vara den absolut kortaste vägen [32]. Detta skiljer sig från A* som genom varje iteration letar efter den kortaste vägen och väljer att fortsätta på denna [33]. D* börjar från slutpunkten och letar sig tillbaks mot startpunkten, här innehar varje nod information om hur långt ifrån målet den befinner sig [34]. Då D* och A* går igenom det fördefinierade området grundligt för att erhålla den mest optimala vägen [33], [34], leder det till en relativt hög behandlingstid jämfört med en RRT algoritm [35].

2.3.1 RRT

RRT bygger upp ett sökträd varefter algoritmen slumpmässigt söker igenom rummet [36], se figur 2.5 nedan. Detta gör att algoritmen snabbt kan få en uppfattning av rummet utan att behöva kartlägga hela miljön. RRT fungerar bra för icke holonomiska problem; där frihetsgraden är lägre än det totala antalet frihetsgrader. Vid användning av RRT för ruttplanering, i en miljö med fasta hinder, utgår algoritmen ifrån en mängd noder som motsvarar öppen yta, samt en mängd noder som utgör hinder. Dessa två mängder är varandras komplement och utgör tillsammans en totalmängd. Vid implementering av algoritmen i olika syften kan totalmängden vara ett fördefinierat område.

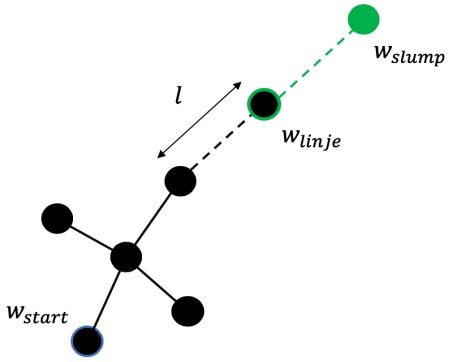


(a) Ett tvådimensionellt RRT-träd som sträcker ut sig i rummet genom att kontinuerligt lägga till nya noder.

(b) Samma RRT-träd i senare skede som har fördelat sig och kartlagt mer eller mindre hela planet den rör sig på.

Figur 2.5

Algoritmen utgår ifrån en given startnod vilken blir den första noden i trädet. Därefter hämtas en slumpmässig nod ur totalmängden. Om noden tillhör mängden av fria noder, så läggs den till som en nod i trädet genom att länkas samman med den nod i trädet som är närmst. Denna typ av sammankoppling begränsas av ett förutbestämt maximalt avstånd. Om avståndet mellan den fria noden och den närmsta noden i trädet skulle överskrida det maximala avståndet, så väljs istället en nod i riktning mot den fria noden, på det maximala avståndet från den närmsta noden. Detta illustreras i figur 2.6. Beteendet fortskrider sedan tills dess att en förutbestämd slutnod är nådd. Det finns ett flertal olika vidareutvecklingar och förbättringar på RRT-algoritmen för olika ändamål.

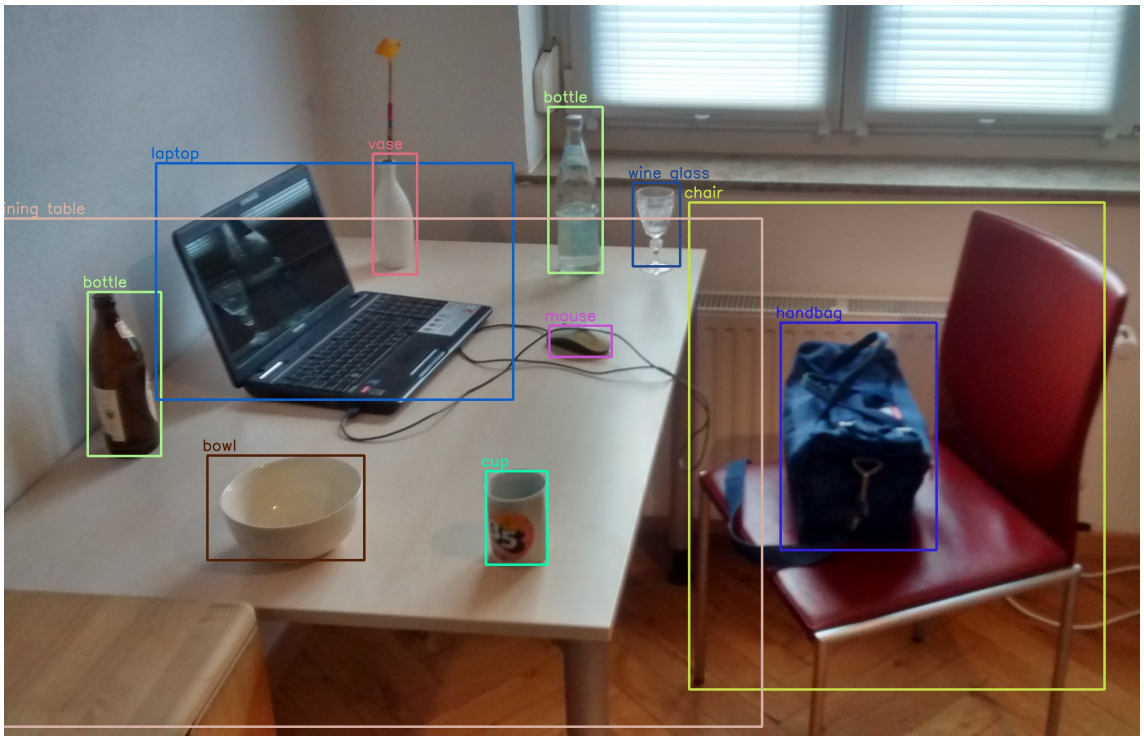


Figur 2.6: Hur sammankopplingen mellan noder i ett RRT-träd bestäms. Trädet har växt fram från en startnod. Den gröna pricken visar en slumpmässig nod från totalmängden av noder, en fri nod. På grund av att w_{slump} är längre bort än det maximala avståndet l så läggs istället noden w_{linje} till i trädet, förutsatt att även denna noden är fri.

2.4 Objektsidentifiering och djup maskininlärning

Objektsidentifiering och klassificering är en vanlig uppgift bland program och system som använder sig av artificiell intelligens, och refererar till beräknande av närvaron eller frånvaron av specifika objekt i bilddata med hjälp av maskininlärning.

Djup inlärning är en metod som använder sig av neurala nätverk för att träna en dator (alt. maskin), och utnyttjas bland annat vid objektsidentifiering [37]. Den vanligaste typen av djup inlärning är övervakad inlärning [38]. Vid objektsidentifiering innebär detta att låta det neurala nätverket behandla ett fördefinierat indataset med bilder, varefter korrektheten i utdatan används för att evaluera hur parametrarna i nätverket skall uppdateras [39]. Dessa parametrar, vilka kallas för vikter, är värden som kan ses som det som styr hur indata-utdata relationen i maskinen fungerar [38]. Vid djup inlärning i objektsidentifieringstillämpningar används annoterade filer vilka innehåller information om vart i respektive bild eventuella objekt som skall upptäckas befinner sig [40]. Detta kan realiseras med gränsboxar, det vill säga rektanglar som målas ut runt objekt för att särskilja dem från omgivningen, se figur 2.7 nedan. I ett djupt nätverkssystem kan det finnas hundratals miljoner justerbara vikter, samtidigt som systemet matas med en liknande storleksordning av annoterade exempel för att uppdatera vikterna och på så sätt lära maskinen [38].



Figur 2.7: Gränsboxar vid användning av YOLO, en välkänd objektsidentifieringsalgoritm. Från [41], CC-BY-SA.

2.4.1 Inlärningsalgoritm vid djup maskininlärning

Felet som beräknas vid en viss evaluering av utdata kallas också för förlust eller kostnad [42]. Den beskrivna träningsprocessen i föregående kapitel sker i de flesta fall via en inlärningsalgoritm baserad runt en metod som kallas för gradient descent [43]. Detta innebär att algoritmen beräknar en gradient som för respektive vikt i nätverket visar vad kostnaden hade ökat eller minskat till om varje vikt inkrementeras med ett litet värde, ofta med en storleksordning mellan $0-1$. Detta lilla värde kallas för learning rate, och är enhetslöst. Därefter uppdateras vikterna i motsatt riktning till gradienten [38], [42]. Därav kan den negativa gradientvektorn ses som en indikation på hur vikterna i nätverket bör uppdateras för att minimera kostnaden och på så sätt uppnå önskat resultat. Gradientvektorn beräknas med hjälp av en algoritm som kallas för backpropagation, vilken grundar sig i kedjeregeln för derivator, alternativt algoritmer som i sig bygger på denna.

2.4.2 Egenskaper som möjliggör realtidsidentifiering

Utnyttjande av Convolutional Neural Networks (CNNs) är vanligast förekommande vid bearbetning av bild– och videodata [44]. CNNs är en typ av djupa neurala nätverk som kräver väldigt lite förbehandling av bilddatan. Anledningen till detta och att realtidsidentifiering och klassificiering av objekt faktiskt är möjligt trots den beräkningsintensiva process som krävs, är främst på grund av den matematiska strukturen som ligger bakom neurala nätverk. Den är parallellt uppbyggd och passar

perfekt arkitekturen hos GPU:er, vilka består av tusentals kärnor optimerade för att behandla flera uppgifter samtidigt. Detta ger en stor fördel vad gäller hastigheten de neurala nätverken klarar av att behandla datan, samtidigt som de är beräkningsmässigt billigare i jämförelse med många andra maskininlärningsmetoder.

2.4.3 Ramverk och algoritmer för objektsidentifiering

Det finns många olika ramverk som kan användas för att implementera objektsidentifiering i flera olika typer av system. Exempelvis utvecklar Apple kontinuerligt Core ML [45] och Vision [46], vilka är två ramverk som kan användas för maskininlärning respektive bildbehandlingsalgoritmer för operativsystemen iOS och macOS.

YOLO är en objektsidentifierings- och klassificieringsalgoritm som kan användas vid mer allsidiga tillämpningar [5]. Namnet står för "You Only Look Once", vilket refererar till att algoritmen bygger på ett neuralt nätverk som genom en enda utvärdering identifierar eventuella objekt och deras klasstillhörighet direkt från fullskaliga bilder. Information om vart i bilden objekten befinner sig genereras, varefter gränsboxar kan målas ut runt objekten i bilden. Algoritmen är baserad på CNN-struktur, inlärningsmetoder för djupa neurala nätverk samt avancerad GPU-teknologi som ökar beräkningshastigheten avsevärt. Den är därför mycket snabb samtidigt som noggrannheten av detekteringar inte påverkas nämnvärt negativt.

Darkflow är ett projekt med öppen källkod som har anpassat YOLO-algoritmen för användning i Tensorflow, vilket är ett populärt maskininlärningsbibliotek, och kan tillämpas för att snabbt och effektivt bygga upp ett objektsidentifieringssystem [47], [48]. Ett användbart verktyg som lätt går att utnyttja tillsammans med Darkflow och Tensorflow är OpenCV (Open Source Computer Vision). Detta är ett datorseende- och maskininlärningsbibliotek med öppen källkod vars mål är att förse användare med verktyg som krävs för att lösa bildbehandlingsproblem med artificiell intelligens [49].

Darkflow kommer med flera färdigtränade modeller och versioner av YOLO. De modeller som kan användas är utvecklade för YOLOv2, även känt som YOLO9000 [50]. Darkflow kommer också med färdigskrivna hjälpklasser som möjliggör att snabbt påbörja träning av en anpassad objektsidentifieringsmodell med funktionalitet för att upptäcka ett godtyckligt antal objekt och klasser.

Vid träning av en anpassad modell i Darkflow kan inledningsvis en förtränad modell användas. Detta innebär att storleken på det dataset som erhållits samt tiden det tar att träna modellen markant minskas. Detta är känt som transfer learning, och innebär att istället för att starta om inlärningsprocessen från början utgå ifrån kända mönster som har lärts in vid lösning av ett annat liknande problem [42].

3

Konstruktion och konfiguration av hybriddrönare

I detta kapitel motiveras de val av koncept och komponenter som ligger till grund för den slutliga produkten av hybriddrönaren. Vidare beskrivs även komponenternas funktioner och sammankoppling sinsemellan. Avslutningsvis presenteras kort drönarens styrsystem och kommunikation samt de flygtester som utförts på drönarens hårdvara.

3.1 Konceptuell design

Valet av den konceptuella designen utgick ifrån de för- och nackdelar som presenterades i avsnitt 2.1.3 om hybriddrönare. På grund av behovet för mekanisk simplicitet och robusthet med hänsyn tagen till stabilitet i luften, ansågs både tiltwing och tailsitter som olämpliga. Med det material som fanns att tillgå bedömdes quadplane som den lämpligaste designen att utgå ifrån.

3.2 Komponentval

Den strukturella designen grundade sig i att återanvända delar från drönare från tidigare projektarbeten för att minska kostnaden för hybriddrönaren. Bland dessa delar valdes en *Raspberry Pi 3b* med tillhörande kamera, en RC-mottagare, ett 3300 mAh batteri (vilket var det batteri med mest energi relativt vikt som fanns), fyra stycken fartreglage (Electronic Speed Control, ESC) samt propellrarna med störst diameter för så energieffektiv flygning som möjligt i enlighet med avsnitt 2.1.2. Det valda batteriet var ett *GensAce 3300 mAh 4S LiPo battery* på 14,8 V, ESCn var *DJI 30A OPTO* och de utvalda propellrarna var *T-motor CF 11x3,7 tums* i kolfiber.

Utifrån lyftkraftsformeln, vilken kan ses i ekvation 2.1 i avsnitt 2.1 är hastigheten i kvadrat och vingarean obekanta. Produkten av dessa bör enligt beräkningar utifrån samma ekvation vara minst 23,8 m^4/s^2 med en lyftkoefficient på 1,4. Utifrån beräkningen ansågs 40 km/h vara ett rimligt val för lägsta kryssningshastighet. Vingarean som behövdes kunde då beräknas till 0,193 m^2. Flygplansdrönaren *Volantex Ranger 2000 757-8* valdes som grund för konstruktionen då denna motsvarade kraven på lyftkraft i FWD-läge och projektets budget. Planet har en vingarea på cirka 0,37 m^2 och väger tillsammans med batteriet 1,45 kg. Utefter detta och tillsammans med

övrigt material och komponenter estimerades en slutlig vikt på cirka 2 kg.

Vidare bestämdes de vertikala motorerna utefter batteriets spänning samt det krav på lyftkraft som ställdes på motorerna med utvalda propellrar. Lyftkraften som krävdes var att kunna hålla drönaren vid konstant altitud vid halv drift och vid full drift kunna lyfta drönarens dubbla vikt [51]. Valet av motorer föll slutligen på *EMAX MT2216-810* kV vilka kunde lyfta 1,01 kg per motor vid full drift och 0,67 kg vid halv drift med propellrar med diameter på 10 tum (1 tum mindre än de valda) samt inte översteg fartreglagets kapacitet.

För att förhålla sig till det övergripande systemet som presenterades i avsnitt 1.4 köptes en flygkontroller av modell *Pixhawk 4* in. Denna komponent valdes då den hade många färdiga funktioner som förenklade utvecklingen av den autonoma drönaren. Dels var Pixhawken utrustad med inbyggda sensorer såsom linjäraccelerationssensor, magnetometer och barometer. Pixhawken inkluderade även en GPS som användes för lokalisering, en krafthanteringsmodul (eng. power management module, PMM) som distribuerade ström till komponenterna och skickade information om batteriets ström- och spänningsnivå samt en signalmodul (eng. auxiliary signal module, ASM) som behandlade styrsignaler från Pixhawken till kontrollytor och driftmotorn (den motor som driver planet i FWD-läge). Förutom de tidigare nämnda komponenterna innefattade Pixhawken även mjukvara som hanterar reglersystemet och autonom flygning. Mängden reglering kunde justeras mellan tre olika nivåer: ingen reglering, höjdreglering och positionsreglering. Mjukvaran beskrivs vidare i avsnitt 5.1.

För att kommunicera med markstationen behövdes en dator på drönaren för att sköta internetanslutningen. Datorn som användes för syftet var Raspberry Pi:n från den tidigare drönaren tillsammans med ett 4G-modem av modell *Huawei E3372h-153* som bedömdes vara tillräckligt liten för att få plats ombord på drönaren. Till datorn kopplades kameran för inspelning av bildström. Hur anslutningen och kameran ställdes in beskrivs vidare i avsnitt 4.1. En extra strömförsörjningskälla (Powerbank) på 2.1 A och 3600 mAh användes för att driva datorn.

Till styrningen som skedde från RC-kontrollern återanvändes en RC-mottagare av modell *WFLY07S* som kopplades till flygkontrollern. Mottagaren kunde dock endast skicka vidare signaler genom PWM (pulsbreddsmodulering) som inte fungerade med flygkontrollern. Detta löstes genom att använda en signalkonverterare av modell *RMILEC V2* som konverterade PWM-signaler från mottagaren till SBus-signaler vilka flygkontrollern kunde tolka.

3.3 Montering och koppling

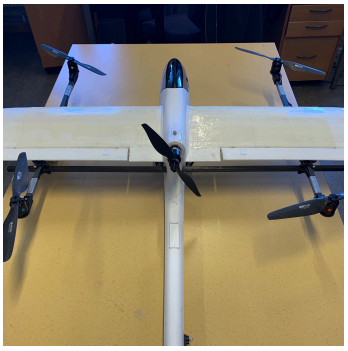
Två kolfiberstänger monterades under vardera av flygplanets vingar. Motorerna monterades med 3D-utskrivna fästen av plast vid båda ändarna av kolfiberstängerna enligt figur 3.1a. För att upprätta stabilitet vid RWD-flygning monterades stängerna parallellt med flygplanets kropp med varje ände i jämn utsträckning från drönarens masscentrum. Monteringen av de extra motorerna resulterade i att vingklaffarna

på flygplanet ej kunde användas. Detta i sin tur medförde inga större problem då vingklaffar oftast används när flygplanet ska lyfta i FWD-läge och denna drönare endast ska lyfta vertikalt.

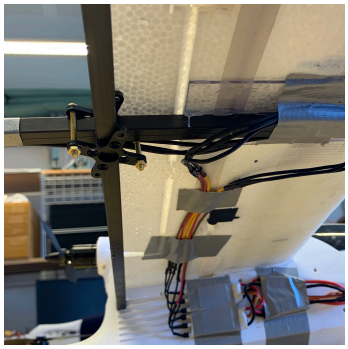
Under tidiga test visades att vingarnas struktur inte var tillräckligt stabil. Vridmomentet som behövdes för att manövrera drönaren, vilket genereras av kraftskillnaden mellan två av motorerna på samma stång, lutade stången och vred vingarna i riktning med pitch-axeln. För att lösa detta monterades en tredje kolfiberstång genom drönarens kropp och fästes samman med de två andra kolfiberstängerna i bakkant av vingen. För att stabilisera vingarna ytterligare applicerades lamineringsepoxi med vävd glasfiber på vingarnas ovansida. Figur 3.1b

ESC:n till de vertikala motorerna monterades längs utsidan på drönarkroppen och kopplades samman med motorerna enligt figur 3.1c. Övrig elektronik kopplades inuti drönaren enligt figur 3.1d. Batteriet fästes kring drönarens masscentrum och kopplades till PMM, vilken sammankopplades med alla fartreglage samt flygkontrollern och försörjer ström till dessa. Fartreglaget som driver driftmotorn kopplades till ASM, vilken i sin tur skickar vidare strömmen till servomotorer som justerar kontollytorna. Raspberry Pi:en samt delarna som inkluderades med Pixhawken kopplades till Pixhawken. Till Raspberry Pi:n kopplades även kameran och 4G-modemet. För utförligare kretsschema, se bilaga B.

(a) Placering av elektronik och kamera



(b) 3D-printat motorfäste



(c) Drönaren efter förstärkningar

(d) Fästen för kolfiberstänger och ESC

Figur 3.1: Slutgiltig konstruktion av drönaren och komponenter.

3.4 Styrsignaler och kommunikation ombord på drönaren

I översikt fungerar drönarens styrsystem genom att de generella navigationssignalerna skickades till flygkontrollern, vilken i sin tur skickade styrsignaler till enskilda motorer och kontrollytor. Signaler från RC-kontrollern konverterades till PWM-signaler av RC-mottagaren, vilka sedan konverterades till SBus-signaler av signalkonverteraren som i sin tur tolkades av flygkontrollern. För kommunikationen mellan Raspberry Pi:en, vilken skötte internetanslutningen för att möjliggöra autonom flygning, och flygkontrollern användes programvaran MAVlink-Router [52]. Denna implementerades på Raspberry Pi:en för att dirigera signalerna via TX/RX pinnar. Flygkontrollern skickar de specifika styrsignalerna i PWM till PMM och ASM, vilka vidaresänds till enskilda motorer och kontrollytor. I bilaga C ses ett schema över kommunikationen mellan komponenterna ombord på drönaren samt även den kommunikation mellan markstationen och drönaren som beskrivs vidare i kapitel **??**.

3.5 Testflygning

Manuella flygtester med stegvis reglering utfördes för att utvärdera konstruktionens funktionalitet i båda flyglägen, där testdatan lagrades av flygkontrollern. Alla tester hann dock inte genomföras på grund av tidsbrist men även de planerade testerna beskrivs nedan. De autonoma flygtesterna förklaras i avsnitt 5.3.

Testerna utfördes i olika stadier. Under första stadiet testades drönarens förmågor i RWD-läge eftersom det tänktes användas som en nödlösning om drönaren i FWD-läge skulle förlora sitt lyft under flygning och tappa höjd drastiskt. Den första förmågan som granskades var att kunna lyfta och hålla sig i luften med sina vertikala propellrar när drönaren styrdes helt manuellt. Detta test utfördes för att granska drönarens mest grundläggande manövrering. Därefter testades drönarens förmåga att reglera sin altitud, vilket behövdes för att visa drönarens stabilitet i höjdled. Detta innebar att drönaren skulle kunna hålla en konstant höjd utan manuell reglering men kunde fortfarande påverkas av exempelvis vind i sidled. Därefter testades drönarens förmåga att reglera sig efter position med hjälp av GPS. Detta innebär exempelvis att drönaren ska kunna behålla sin position om ingen indata ges även vid vind. Positionsläget innefattade även höjdreglering. Syftet med detta test var att visa att drönaren på egen hand kunde navigera efter koordinater i det tredimensionella rummet vilket i förlängningen innebar att drönaren borde kunna följa en bestämd rutt. Därefter utfördes tester som granskade drönarens prestanda i RWD-läge, så som maxhastighet, maximal stigningshastighet, maximal flygtid och maximal flyglängd.

Efter testerna på drönarens prestanda i RWD-läge skulle drönaren testas i FWD-läge. Alla tester i FWD-läge hanns inte med på grund av tidsbrist men även de som inte hanns med presenteras nedan. I det initiala testet i detta stadie testades

drönarens förmåga att övergå från RWD-läge till FWD-läge, hålla stabil altitud vid rak flygning under ett flertal sekunder, och sedan övergå tillbaka till RWD-läge. Därefter skulle samma test utföras igen men denna gång tillsammans med en 180 graders sväng åt både vänster och höger samt en stigning och en sänkning. Vid detta test lyfte drönaren i RWD-läge, övergick till FWD-läge och började cirkulera runt operatören för att sedan oavsiktligt nå den kritiskt lägsta höjden om 15 meter där drönaren var programmerad att automatiskt övergå tillbaks till RWD-läge. Detta tillsammans med den höga hastigheten gjorde att operatören tappade kontrollen över drönaren och kraschade. Vidare tester kunde därefter inte utföras. Dessa tester hade innefattat altitud- och positionsreglering av samma skäl som motsvarande tester i RWD-läge. Slutligen hade prestandatester i FWD-läge utförts för att samla ytterligare statistik på drönaren.

4

Trådlös överföring av information, mätdata och realtidsström

För att det skall vara möjligt att både få drönaren att flyga autonomt och kunna implementera objektsidentifiering så krävs det en trådlös överföring av information mellan drönaren och markstationen. Detta görs genom att erbjuda fjärrtjänster och tillhandahålla en uppkoppling mot internet.

4.1 Nätverkskrav från delsystem på drönaren

Den tilltänkta systemlösningen, av vilken en överblick ges i kapitel 1.4, skapar efterfrågan av fjärrtjänster enligt följande bakgrunder:

- Drönarens flygkontroller låter sig styras av anslutningar till en markstation med förspecificerad IP-adress där det körs en särskild mjukvara vid namn QGroundControl. QGroundControl beskrivs mer i avsnitt 5.1. För att QGroundControl skall kunna styra drönaren krävs en stabil anslutning till den givna IP-adressen.
- Drönaren producerar bilddata som skall överföras i ett särskilt format på ett sådant sätt som tillåter mottagaren att bearbeta datan. Det kräver en stabil anslutning till mottagaren.

Att använda sig av 4G framför till exempel WiFi tillåter drönaren att få internetuppkoppling utanför WiFi-nätets räckvidd. Med ovanstående som bakgrund valdes ett 4G-modem som uppkopplingslösning till internet. I Sverige är 4G-täckningen bra och utbredd på många platser [53]. En 4G-uppkoppling erbjuder teoretiska hastigheter på upp till 300 Mbit/sekund till skillnad från en 3G-uppkoppling som har en teoretisk högsta hastighet på 32 Mbit/sekund [54]. Då bilddata generellt sett innefattar större volymer data, ansågs 4G som det lämpligare valet.

UDP och TCP är två protokoll som HTTP och RTSP kan användas över. TCP-protokollets handskakning skapar inte problem eftersom att 4G-anslutningen är så pass snabb som den är. Att använda sig av HTTP framför RTSP tillåter en smidigare hantering över webben genom en webbläsare utan att särskild mjukvara behöver installeras.

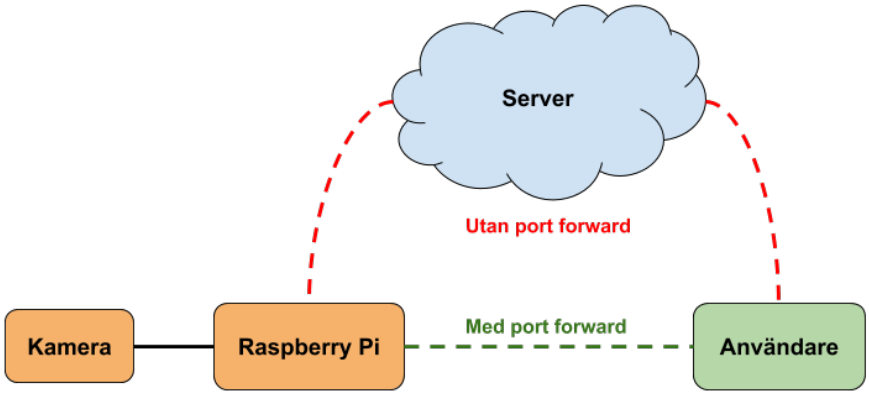
4.2 Skicka data för extern bearbetning

För att samla in data att skicka användes en kamera. Kameran anslöts till en Raspberry Pi som körde open-source kameraprogrammet RPi Cam Web Interface [55]. Kameraprogrammet skapar en videoström eller enstaka bilder utifrån en ansluten kamera och skapar ett webbgränssnitt att använda för konfiguration och visning. Denna mjukvara kräver en särskild webbserver.

En webbserver av typen NGINX ställdes in på Raspberry Pi och ett domännamn konfigurerades för användning med webbservern. Det installerades även PHP på servern så att den kunde köra kameraprogrammet och de utvecklade mjukvarorna. Servern kunde erbjuda webbtjänster och bearbeta inkommande anslutningar. Detta tillät en annan dator att ansluta sig till Raspberry Pi:n för att strömma video i realtid.

Med hjälp av kameramjukvaran kunde därför videoströmmen hämtas från Raspberry Pi varifrån klienten (objektsidentifieringsprogrammet) sedan kunde bearbeta denna för inläsning. Raspberry Pi och klienten var då direkt anslutna till varandra enligt den gröna vägen i figur 4.1. Till denna lösningen skapades några mjukvaror för övervaka drönarens IP-adress, dessa beskrivs i avsnitt 4.3.

Denna lösningen fungerade utan anmärkning i den kontrollerade miljön som produkten togs fram i. Vid övergång till den riktiga körmiljön med ett 4G modem så uppstod ett oväntat problem som berodde på modemets hårdvara beskrivet i avsnitt 4.5.



Figur 4.1: Videoströmmning från Raspberry Pi med och utan port-forwarding-möjligheter. De streckade linjerna motsvarar nätverksanslutningar medan fulla linjer motsvarar fysiska kopplingar.

4.3 Stödprogramvara för IP-adressering och bildtagning

Under arbetets gång utvecklades följande mjukvaror: (1) en som uppdaterar IP-adresser, (2) en som tillhandahåller IP-adresser, (3) en som tar en bild och skriver till en fil samt (4) en som tar en bild och skriver till en fil på en fjärrserver.

Den första mjukvaran byggdes i Java. Programmet uppdaterar drönarens IP-adress och håller en tredje server uppdaterad. På så vis är drönarens aktuella IP-adress alltid känd. Syftet med mjukvaran är att kunna nå drönaren över internet och då behövs dess aktuella IP-adress.

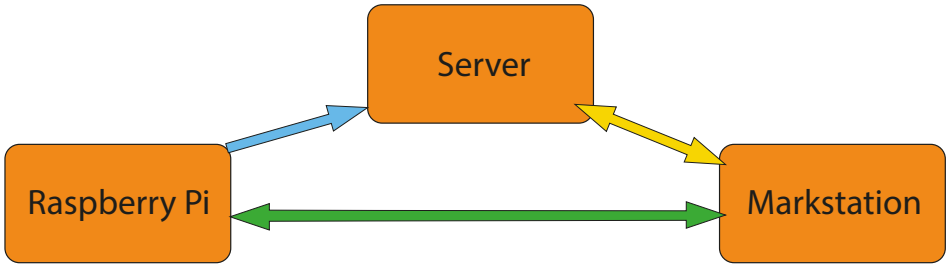
Den andra mjukvaran byggdes i PHP och sparar data i en MySQL-databas. Programmet har funktionalitet för att kunna tag emot en IP-adress, lagra den samt ge tillbaka den i JSON-format. Syftet med mjukvaran är att tillfälligt lagra drönarens dynamiska IP-adress på en server med statisk IP-adress för att alltid kunna nå drönaren.

Den tredje mjukvaran är egentligen ett bash-skript och tar en bild med det inbyggda programmet *raspistill* och skriver det till en fil samt upprepar detta med givna mellanrum. Syftet med mjukvaran är att kunna hämta bilder från den givna platsen.

Den fjärde mjukvaran är också ett bash-skript och tar en bild med det inbyggda programmet *raspistill* och skriver det till en fil på en fjärrserver med programmet SCP (Secure Copy) samt upprepar detta med givna mellanrum. Syftet med mjukvaran är att kunna ta bilder och lägga dessa på en extern server så att dessa kan nås från en annan plats när drönaren inte kan nås.

4.4 Inställning av extern server och mellanhand

För att hantera drönarens dynamiska IP-adress användes en extern server med statisk IP-adress för att hålla reda på drönarens aktuella IP-adress med hjälp av de mjukvaror som utvecklades och beskrevs i avsnitt 4.3. På servern kördes NGINX, PHP och MySQL. Den externa serverns uppgift var att ta emot IP-uppdateringar från drönaren och tillkännage IP-adresser på förfrågan från de som önskade ansluta till drönaren enligt figur 4.2.



Figur 4.2: Den ursprungliga nätverkslösningen som inte fungerar med hårdvara som har beskriven begränsning. Den blå pilen respresenterar Raspberry Pi:en som ger sin IP-adress till servern. Den gula representerar när markstationen vill hämta IP-adressen för Raspberry Pi:en från servern. Den gröna är när markstationen begär data från Raspberry Pi:en.

4.4.1 Den externa serverns gränssnitt mot internet

Vid utveckling av mjukvarorna togs det särskild hänsyn till att skapa ett smidigt gränssnitt att använda vid utveckling av annan mjukvara. För att göra det smidigt att skicka in ny data användes GET som metod och JSON för formatet att tag emot svaret i. Nedan visas två exempel på hur enkelt det är att interagera med mjukvarorna över internet.

Vid uppdatering av drönarens IP-adress användes följande URL:

```
GET: https://www.rorgren.se/kandidat/update.php?setDroneIP=4.3.2.1
Svar vid framgång: {"status": "success"}
```

Vid kontroll av drönarens IP-adress användes följande URL:

```
GET: https://www.rorgren.se/kandidat/retreve.php
Svar vid framgång: {"server": "0.0.0.2", "drone": "4.3.2.1"}
```

Det uppstod ett problem med det inköpta modemet, beskrivet i avsnitt 4.5. Av denna anledning gjordes denna server om. Serverns nya syfte blev att agera som en mellanhand för alla anslutningar och vidarebefordra dessa till annan ansluten klient enligt figur 4.3.

Figur 4.3: Nätverkskopplingen efter lösningen för hårdvarubegränsnignen. De blå pilarna representerar den initiala anslutningen som markstationen och Raspberry Pi:en måste göra till mellanhanden. Därefter kan mellanhanden ta emot data från Raspberry Pi:en och skicka den vidare till markstationen.

4.5 Blockering av inkommande anslutningar

Vid utveckling och utvärdering av mjukvaran för bland annat strömning kördes detta i en kontrollerad nätverksmiljö. Detta innebar att hårdvarubegränsningar och andra hinder kunde undvikas genom att konfigurera nätverket på ett annat sätt.

Vid inköp av 4G-modemet förutsattes det att modemet skulle fungera efter de förutsättningar som beskrevs i 4.1. Dessvärre fungerade inte modemet som tänkt. Raspberry Pi:en som var inställd som en server vars syfte var att acceptera inkommande anslutningar. Den fick inga att behandla eftersom att modemet blockerade dessa. I ett nätverk med en router konfigureras normalt routern att skicka vidare inkommande anslutningar till en särskild enhet i nätverket. Detta är något modemet inte hade stöd för och ledde till att Raspberry Pi:en inte kunde ta emot inkommande anslutningar med just det modemet.

För att lösa problemet fanns två alternativ. Det första vara att köpa in ett annat modem och det andra var att ställa in en mellanhand som Raspberry Pi får ansluta sig till och vidarebefordra alla överföringar till en annan klient. Det senare alternativet valdes eftersom att de modem som fanns tillgänglig på marknaden med de eftersökta funktionerna var större och inte fick plats på drönaren. Det senare alternativet ledde också till att de tre första mjukvarorna, beskrivet i avsnitt 4.3, inte längre var nödvändiga för projektet eftersom att inga inkommande anslutningar behövde göras till drönaren.

5

Autonom flygning

Att få en hybriddrönare att flyga autonomt omfattar ett antal olika steg. Denna uppdelning kan göras enligt:

- Start
- Övergång mellan flygplans- och rotorläge
- Flyga mellan vägpunkter
- Kollisionsundvikning
- Landning

Den flygkontroll som köptes in för att lösa reglersystemet kan även användas för att lösa fyra av punkterna ovan genom nedladdning av en fast programvara till den valda flygkontrollern. Programvaran hanterar dock inte kollisionsundvikning, utan detta behöver då utvecklas för hybriddrönaren. Därmed behandlar detta kapitel först de delar som löses med hjälp av flygkontrollern, följt av ett delsystem till kollisionsundvikningen som utvecklas i detta projekt och slutligen beskrivs de flygtester som utförts för att utvärdera den autonoma flygförmågan. Kollisionsundvikningen hann aldrig implementeras och därför testades endast den autonomi som flygkontrollern hanterar.

5.1 Autonom flygning med flygkontroll

Flygkontrollern av modell *Pixhawk 4* introducerades i avsnitt 3.2. Den mjukvara som kort presenterades innefattar ett nedladdningsbart program, *PX4*, som sköter regleringen av drönaren i båda flyglägen samt även navigeringen mellan vägpunkter. Som komplement till detta användes programmet *QGroundControl* för att definiera rutt samt övervaka flera av drönarens mätvärden. Programmet användes av operatören vid markstationen och kommunicerade via internet till 4G-modemet ombord på drönaren.

5.2 Kollisionsundvikning

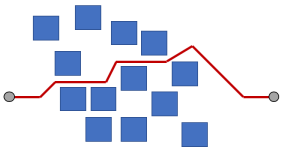
Undersökningar påbörjades initialt för hur kollisionsundvikning ska ske för en drönare i flygplansläge. På grund av drönarens höga hastighet samt begränsad vikt och utrymme för ytterligare komponenter ansågs det ej vara möjligt att implementera dynamisk hinderundvikning i realtid med hjälp av sensorer. Arbetet gick då istället vidare med att undersöka ruttplaneringsalgoritmer för kända miljöer där hänsyn kan tas till statiska, kända hinder. För att drönaren ska kunna flyga helt autonomt, utan risk att krocka med hinder, är troligvis en kombination av dynamisk och statisk hinderundvikning nödvändigt.

5.2.1 Utveckling av ruttplaneringsalgoritm

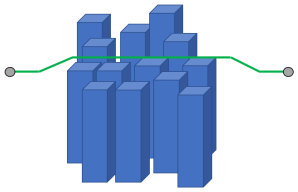
Då drönaren flyger i rummet krävs en algoritm som kan hantera hinder i en tredimensionell miljö, om den skall kunna tillämpas på detta projekt. RRT (se beskrivning i kapitel 2.3.1) är en ruttplaneringsalgoritm som ofta dyker upp i samband med drönare. Varianter av RRT som fungerar i tre dimensioner har utvecklats tidigare.

Det finns en variant av en RRT-algoritm som är anpassad just för FWD i hindertäta miljöer, till exempel större städer [56]. I denna algoritm förenklas det tredimensionella problemet genom att projicera startpunkt, målpunkt och hinder på ett horisontellt plan, varefter den vanliga RRT-algoritmen istället kan utföras i två dimensioner. En sådan algoritm kan i verkligheten anpassas så att den kan utnyttja tredimensionella kartor över städer för att planera vägen för en drönare innan flygningen är påbörjad. Denna variant av RRT ansågs likna behovet och appliceringsområdet för detta arbete, och ansågs därför som en passande algoritm att utgå från.

Motiveringen till att algoritmen görs i två dimensioner är att en FWD är betydligt mer manövrerbar horisontellt än vertikalt [56]. En RRT i planet ger också en snabbare och mindre komplex lösning än i rummet. Att endast flyga i ett plan kan dock resultera i en mindre effektiv bana för drönaren om den flyger in i ett område med många täta hinder. I många fall hade därför en snabbare och mer effektiv rutt kunnat väljas om drönaren avvärjde från dess förutbestämda höjd och istället flög några meter högre som illusterat i figur 5.1.



(a) Miljö sedd uppifrån. En väg i två dimensioner.



(b) Miljö sedd från sidan. En väg som tillfälligt byter höjdled.

Figur 5.1: De två bilderna jämför hur rutten för drönaren blir om den håller samma höjd och flyger runt hinder jämfört med att flyga över dem.

I Sverige är den genomsnittliga bebyggelsen såpass låg att en rutt för en drönare kan förenklas genom att flyga över de flesta hinder istället för att flyga runt. För att implementera detta önskvärda beteendet på en drönare beslutades att skapa en ny algoritm med den tidigare algoritmen som grund [56], med skillnaden att drönaren vid behov skulle kunna lämna det förutbestämda planet och på så vis flyga över eventuella hinder.

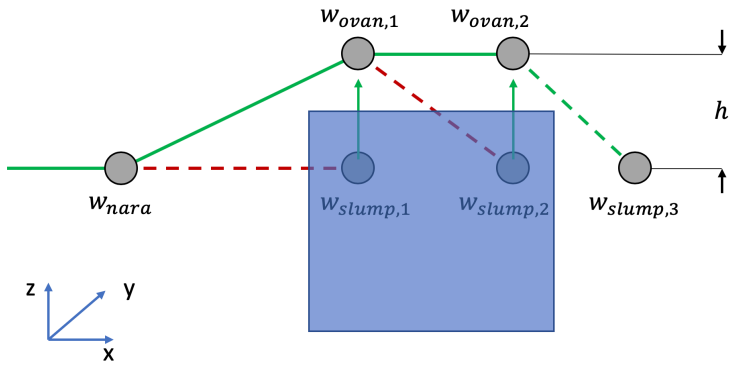
5.2.2 Framtagen algoritm

Den ursprungliga algoritmen som legat till grund för detta arbete benämns som RRT_{plan} [56]. Den vidareutvecklade algoritmen som framtagits under projektets gång, vars syfte var att effektivisera ruttplaneringen för drönaren, benämns för RRT_{lyft} och kan ses i algoritm 1 nedan.

Algorithm 1 RRT_{lyft}

1: Välj ett horisontellt plan mellan startpunkten och målpunkten inom intervallet 10-120
meters höjd.
2: Projicera start och målpunkt i planet, w_{start}, w_{mal}. Lägg till dessa i trädet.
3: Välj en slumpmässig punkt w_{slump} i samma plan (i riktning mot w_{mal}).
4: Välj den besökta noden i trädet w_{nara}, som är närmast w_{slump}.
5: Välj en punkt w_{linje} på ett förutbestämt avstånd från w_{nara} på linjen mot w_{slump}.
6: Kolla om vägen mellan w_{slump} och w_{nara} korsar ett hinder.
7: Om ingen kollision, lägg till w_{linje} i trädet.
   (a) annars, kolla om en punkt ovanför, w_{ovan} ligger i ett hinder
   (b) om ingen kollision, lägg till w_{ovan} i trädet
   (c) annars gå tillbaka till punkt 3
8: Fortsätt sträcka ut mot w_{slump}.
9: Fortsätt med steg 3-8 tills w_{mal} är med i trädet.
10: Hitta hela vägen från w_{start} till w_{mal}
11: Beräkna en väg med rimlig svängradie för verklig flygning

I figur 5.2 visualiseras steg 7 i algoritmen som är den väsentliga skillnaden mellan RRT_{lyft} och RRT_{plan}. Om vägen mellan $\text{w}_{slump,1}$ och w_{nara} ligger i ett hinder, kontrolleras en nod w_{ovan} på ett fixt avstånd $\Delta\text{z} = h$ ovanför $\text{w}_{slump,1}$. Om den noden också ligger i ett hinder kasseras båda noderna och algoritmen börjar om från steg **3**. Om w_{ovan} inte ligger i ett hinder läggs istället w_{ovan} till i trädet. Då lämnas alltså planet tillfälligt. Nästa punkt $\text{w}_{slump,2}$ befinner sig också i hindret och byts på samma sätt ut mot $\text{w}_{ovan,2}$. Nästkommande punkt $\text{w}_{slump,3}$ ligger utanför hindret och vägen mellan $\text{w}_{ovan,2}$ (nya w_{nara}) och $\text{w}_{slump,3}$ korsar inget hinder så noden $\text{w}_{slump,3}$ läggs till i trädet och algoritmen återgår till steg **3** och slumpar en ny punkt.

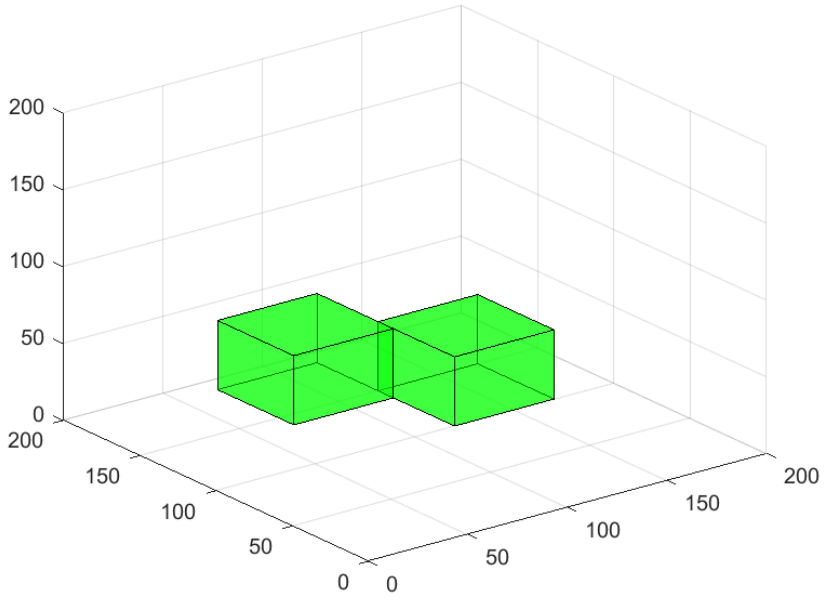


Figur 5.2: Steg **7** i algoritmen då noderna $\text{w}_{slump,1}$ och $\text{w}_{slump,2}$ ligger i ett hinder. De röda streckade linjerna representerar vägar som inte går att ta på grund av att de korsar hinder. De heldragna gröna linjerna representerar vägen som lagts till i trädet. Den gröna streckade linjen representerar en väg som går att gå.

5.2.3 Simuleringar av ruttplaneringsalgoritm

För att testa ruttplaneringsalgoritmen gjordes simuleringar i MatLab vilka utgick från en open-source kod på MathWorks [57]. Den färdiga koden skapade en variant av RRT som skrevs om för att anpassas till projektets syfte, se bilaga G för fullständig kod.

Två olika simuleringar gjordes. Den ena använde sig av en RRT-algoritm i planet, med hinder som definierats i 3D, för att simulera en algoritm liknande RRT_{plan}. Den andra simuleringen gjordes i syftet att simulera en RRT som även kan röra sig över hinder, RRT_{lyft}. För att ta reda på vilken av algoritmerna som var fördelaktig jämfördes sedan skillnader i sträcka för de båda algoritmerna.



Figur 5.3: Simuleringsmiljö i MatLab med två låga hinder

För att utföra simuleringarna skapades ett rum i Matlab med storlek 200^3 volymenheter. I detta rum definierades noder som heltalskoordinater. Den totala mängden noder i rummet uppgår därmed till 200^3 stycken. Simuleringarna utfördes i en miljö med två låga hinder. Med lågt hinder syftas här på ett hinder vars höjd är ett kort avstånd ovanför det horisontella plan som RRT-trädet ursprungligen rör sig i. Simuleringar i mer komplexa miljöer kunde inte genomföras på grund av begränsad beräkningskraft och begränsad tid för att effektivisera koden.

5.2.4 Implementation av ruttplaneringsalgoritm

Initialt genomfördes studier på hur ruttplaneringsalgoritmen skulle kunna implementeras på drönaren med Python. Python är beräkningsmässigt mer krävande än vissa andra programmeringsspråk, men detta ansågs inte vara ett problem då beräkningarna kan göras på en extern dator. På grund av tidsbrist ansågs dock inte en slutgiltig implementering vara möjlig. Därför övergavs Python och denna del av projektet gick helt över till utvecklandet av själva algoritmen samt simuleringar av denna i MatLab.

5.3 Autonom testflygning

För att utvärdera den autonoma flygförmågan gjordes tester att följa en rutt i tre dimensioner. Då ruttplaneringsalgoritmen aldrig hann implementeras kunde inte denna eller någon form av hinderundvikning testas. På grund av tidsbrist testades inte heller autonom flygning i FWD-läge eller autonom övergång mellan FWD och RWD-läge. De två testerna utfördes alltså endast i RWD-läge och undersökte avvikelserna från rutt samt stabilitet under flygningen. Vägpunkter sattes ut med hjälp av *QGroundControl* vilka drönaren skulle följa i tur och ordning. Här användes mjukvaran *PX4* för att navigera mellan vägpunkterna.

6

Objektsidentifiering för detektering av människor

Objektsidentifieringssystemet som utvecklades grundade sig i att lämna över den beräkningsintensiva bildbehandlingen till markstationen. Detta innebar att en bildström från kameran på drönaren skulle överföras till markstationen, varefter objektsidentifieringen behövde ske innan de färdigprocesserade bilderna kunde renderas för användaren. Detta kapitel går igenom utvecklingsprocessen för detta system under projektets gång.

6.1 Förarbete och val av objektsidentifieringsalgoritm

Under förarbetet och undersökningen av hur objektsidentifieringen skulle implementeras testades initialt två tillvägagångssätt parallellt, varefter det mest lämpade användes. Det ena tillvägagångsättet grundade sig i användandet av Core ML och Vision, vilka beskrivs i kapitel 2.4.3. Eftersom Core ML och Vision för närvarande enbart kan användas på Apples operativsystem togs beslutet att inte fortsätta undersöka och utveckla ett identifieringssystem med hjälp av dessa ramverk. Istället vidareutvecklades ett tillvägagångssätt som bygger på YOLO-algoritmen med Darkflow, vilka också beskrivs i kapitel 2.4.3, då detta ansågs ge ett identiferingssytem med fler anpassningsmöjligheter. Dessutom är YOLO i sig mycket snabbare främst tack vare sitt CUDA-stöd, vilket lämpade sig bra då modellen skulle kunna ta emot en videoström i realtid.

Det fanns två typer av färdigtränade modeller med Darkflow tillgängliga för nedladdning: modeller tränade på datasetet PASCAL VOC [58] och dess data mellan 2007 och 2012, samt datasetet COCO: Common Objects in Context [59]. Tester med modeller tränade på det förstnämnda dataset genomfördes på bilddata fångat av drönare. De färdiga modellerna visade sig fungera mycket undermåligt i detta fall, då datasetet till största del inte innehåller bilder med människor tagna från samma vinkel drönaren har gentemot marken. Båda dataseten innehåller liknande bilddata,

vilket gjorde att modeller tränade på det sistnämnda datasetet inte behövde testas då även dessa med största sannolikhet skulle lyckas dåligt med att detektera människor från flygbilder. Resultatet innebar att en anpassad modell behövde tränas, vilket i sig krävde ett dataset innehållande bilder med liknande karakteristik som den bilddata som i slutändan skulle tas av drönarens kamera. Dessutom behövde respektive bild i detta dataset även innehålla personer för att objektsidentifieringssystemet skulle kunna användas i enlighet med syftet.

6.2 Framtagning av dataset

En undersökning gjordes för att hitta lämplig bilddata till träning av det neurala nätverket. Tillslut användes videofilmer från datasetet Okutama-Action [60]. Då detta dataset innehöll annoteringsfiler med klasser som ej var önskvärda och dessutom var i fel filformat för applicering i Darkflow, användes enbart videofilmerna och ett eget dataset skapades utifrån dem.

Darkflow behandlar en bild i taget, vilket innebar att specifika bilder behövde extraheras från videodatan. Detta gjordes med multimediaspelaren VLC Media Player [61]. Det antogs att videodatan som senare skulle fångas av kameran på drönaren skulle ha en upplösning på 1920x1080 pixlar, vilket påverkade beslutet att konvertera videodatan från datasetet till detta format. Sedan extraherades var 50:e bild från videodatan, vilket resulterade i en samling av 751 bilder. Två av videofilmerna i datasetet användes inte till detta, utan sparades istället undan tills vidare i syfte att senare kunna testa det tränade neurala nätverket med dessa.

Generering av de annoteringsfiler som krävdes för träningen utfördes med LabelImage som är ett grafiskt verktyg för att annotera bilder med hjälp av gränsboxar [40]. Verktyget har öppen källkod och kan generera annoteringsfiler i PASCAL VOC-format, vilket är det format som YOLOv2 använder i Darkflow. Bland de 751 bilderna i datasetet märktes 4722 instanser av personer totalt med hjälp av LabelImage.

6.3 Träning och implementering av anpassad modell

Några enstaka förändringar i den ursprungliga modellen krävdes för att använda sig utav Darkflows färdigskrivna träningsprocess, vilka utfördes enligt dokumentationen på Github-sidan för Darkflow [47]. Vidare skrevs också ett par korta scripts i Python för att underlätta arbetsflödet och anpassning av träningen.

För att minska den nödvändiga storleken av träningsdatan, och utnyttja transfer

learning-förmågan hos Darkflow (läs mer om transfer learning i kapitel 2.4.3), utgick den anpassade modellen ifrån en förtränad modell som kallas för Tiny Yolo [62]. Detta är en minivariant av YOLO med färre lager än det ursprungliga YOLO-nätverket. Modellens struktur resulterar i sämre precision, men i gengäld kan den behandla bilder i mycket högre hastighet. Detta beslut togs med hänsyn till att modellen var tvungen att kunna behandla realtidsvideo.

Den anpassade modellen tränades därefter att upptäcka en enda klass, benämnd "person". En mer ingående beskrivning av träningsprocessen kan erhållas i appendix A. Den ansågs sedan uppnå godtagbara resultat när den till viss del kunde identifiera människor på bilder som testades. Denna testning utfördes genom att implementera modellen i ett Python-script där OpenCV användes för att behandla det extraherade resultatet som det neurala nätverket genererat. Med hjälp av detta bibliotek kunde bilder från en videoström via en fil eller webkamera på datorn, eller via en server med livevideo-data, läsas in till det neurala nätverket, varefter gränsboxar ritades ut i bilder där personer upptäckts. Därefter kunde de färdigbehandlade bilderna renderas i tur och ordning. Python-scriptet utökades sedan med funktionalitet för att spara den renderade videon som en fil på datorn.

Efter att kontinuerlig inläsning av bilder från en existerande videofil eller webbkamera kopplad till datorn lösts, krävdes funktionalitet för att ersätta denna bilddata med videoströmmen från drönaren som skulle skickas över ett nätverk. Även detta kunde göras med hjälp utav OpenCV, vilket implementerades när strömmingsprogrammet hade utvecklats. De python-script som används för objektsidentifering kan erhållas i appendix F.

7

Resultat

I detta kapitel presenteras en övergripande bild av prototypen som detta arbete lett fram till samt de mer ingående resultaten av de olika delmomenten som behandlas i kapitel 3, 4, 5 och 6. Inledningsvis presenteras en överblick och sedan presenteras de flygtester som utförts på drönaren samt resultatet angående drönarens egenskaper, prestanda och effektivitet i RWD- respektive FWD-läge. Vidare beskrivs resultatet från simulering av ruttplaneringsalgoritmer i MatLab och utfall av objektsidentifieringsmodell för detektering av människor.

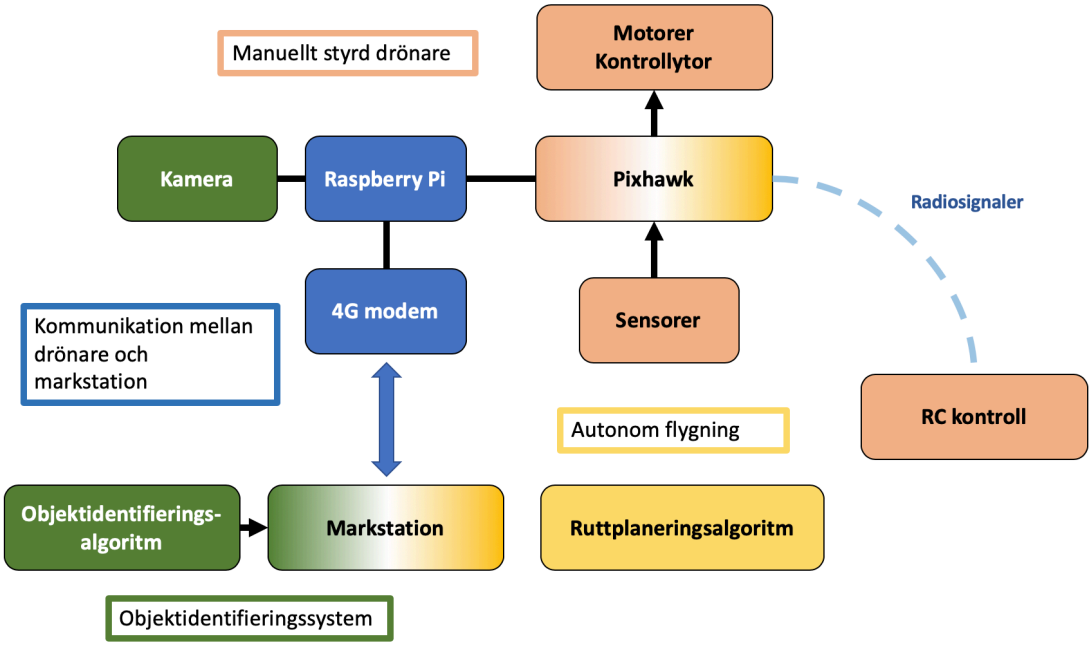
7.1 Övergripande resultat av slutprodukt

Arbetet presenterat i denna rapport har lett till en hybrid mellan en FWD och en RWD som besitter såväl VTOL-egenskaper som förmåga att glidflyga som ett flygplan. Drönaren kan även starta, landa och flyga mellan vägpunkter autonomt. Ett objektidentifieringssystem för att identifiera människor vid sök- och räddningsuppdrag samt en ruttplaneringsalgoritm för statisk hinderundvikning har även tagits fram.

En övergripande bild av det slutgiltiga systemet visas i figur 7.1 där det går att se hur de olika delsystemen hänger ihop. De sandfärgade boxarna beskriver de komponenter som behövs för att hybriddrönaren ska kunna flyga via manuell styrning. De gröna modulerna krävs för objektidentifieringssystemet medan de gula boxarna indikerar de delar som behandlar den autonoma flygningen. Slutligen beskriver de blå boxarna kommunikationen mellan drönaren och markstation.

Ruttplaneringsalgoritmen är inte sammankopplad då denna aldrig implementerades. Flygkontrollern är både sandfärgad och gul då den dels sköter reglersystemet som behövs för den manuella flygningen, men även innehar programvara som hanterar autonom start, landning och flygning mellan vägpunkter. Markstationen är både gul och grön då den både används för att applicera objektidentifieringen, men också skickar uppdrag för den autonoma flygningen som utförs av Pixhawken. En fungerande lösning för realtidsvideoströmning färdigställdes inte inom tidsramen för

projektet vilket resulterade i att objektidentifieringen inte kunde appliceras i realtid.



Figur 7.1: Övergripande bild av det slutgiltiga systemet för sök- och räddningsuppdrag. De sandfärgade blocken representerar komponenterna för den manuella styrningen av drönaren. De gulmarkerade blocken i representerar den autonoma flygningen. Gröna block är relaterade till objektsidentifieringen och blått representerar kommunikationen mellan drönaren markstationen.

7.2 Resultat från flygtester

I detta delkapitel görs en genomgång av resultaten från samtliga flygtester som beskrivs i kapitel 3 och 5, såväl manuella som autonoma i både FWD- och RWD-läge. Endast den data som anses relevant för respektive flygtest presenteras. I bilaga D ses utförligare testloggar.

7.2.1 RWD med altitudreglering

Vid test i RWD-läge med höjdreglering och manuell styrning via RC-kontroller syns flertalet brister i regleringen. Drönaren har en tendens att vrida (yaw) åt vänster, sett ifrån drönarens normala färdriktning, samt rotera uppåt (pitch) vilket resulterade i att drönaren ville backa. Detta reglerades manuellt med RC-kontrollern utan större problem men upplevdes osäkert. På grund av osäkerheten avslutades testet efter ungefär 2 minuter då testet hade låg prioritet.

7.2.2 RWD med positionreglering

I RWD-läge med positionsreglering och manuell styrning med RC-kontroller kunde drönaren följa inmatningar utan någon tydligt avvikelse. Vid 50% gasreglering (eng. throttle) och utan övriga inmatningar från RC-kontrollern kunde drönaren sväva på konstant position och höjd. Vid 0% gasreglering sjönk drönaren sakta med konstant hastighet. Drönarens maximala stigning under testet var 10 km/h till en höjd på 29 m ovan mark enligt flygdata. I horisontellt led var drönarens högsta hastighet 26,6 km/h och den färdades totalt en sträcka på 563,8 m. Drönaren flög i olika riktningar tills batteriet tog slut vilket resulterade i en flygtid på 6 minuter och 9 sekunder. När batterinivån blev låg blev drönaren svårstyrd och kunde bete sig oförutsägbart, vilket var genomgående för alla tester.

7.2.3 Flygtid i RWD

Drönarens flygtid i RWD-läge testades även genom att manuellt flyga drönaren i cirklar med en hastighet på cirka 5 km/h. Testet pågick från fullt batteri tills strömförsörjningen inte längre räckte för att hålla drönaren i luften. Vid detta flygtest blev flygtiden cirka 6 minuter och 30 sekunder. Flygsträckan under testet var 526,2 meter.

Vid det tredje och det fjärde testet, som utfördes i syfte att testa positionsreglering och uthållighet, hade drönaren problem att ställa in sig i RWD-läge med positionsreglering. Det dröjde ungefär 2,5 respektive 3 minuter för drönaren att ställa in sig för positionsreglering.

7.2.4 RWD med autonom navigering

Vid det första försöket att testa autonom navigation mellan vägpunkter var drönarens flygning instabil vid svängar, vilket gjorde att fortsatt flygning med autonom ruttföljning ansågs osäkert. Därför avbröts testflygningen för att landa drönaren manuellt. Flygdatan visade att drönaren i två fall tog abnormala rutter mellan vägpunkterna. Granskning av testlogg visar att både rotation och rotationshastighet av pitch respektive roll oscillerade. Av rutten som genomfördes visar loggen en högsta positionsavvikelse på cirka 4 m i horisontellt plan. Testloggen visar också att positionsbörvärderna avvek från den ideala raka sträckan mellan vägpunkterna som förväntades.

Under det andra testet för ruttföljning via vägpunkter lyckas drönaren följa den förbestämda rutten bättre. I övrigt betedde sig drönaren likt tidigare, med fortsatt hög instabilitet i svängar och med oscillation av rotation och rotationshastighet. Den högsta positionsavvikelsen var även här 4 m men genomsnittligt lägre än första. Börvärdena avvek återigen stundtals från den ideala rutten.

7.2.5 Övergång mellan RWD- och FWD-läge

Vid det första testet i FWD-läge lyftes drönaren med manuell flygning i RWD-läge till en altitud på cirka 50 meter, där en lyckad övergång till FWD-läge genomfördes vid cirka 30km/h. Under övergång sänktes drönarens altitud med cirka 1.9 meter. Testet fortsatte med en rak flygning i 16 sekunder, därefter utfördes en lyckad övergång tillbaka till RWD läge vid ungefär 40km/h. Vid övergång tillbaka till RWD-läge visades kraftiga oscilleringar i rotationer och altituden minskade hastigt under 20 sekunder tills drönaren kraschlandades. Ingen skada märktes dock på drönaren efter testet.

7.2.6 Manövrerbarhet i FWD-läge

Vid det andra testet i FWD-läge lyftes drönaren med manuell flygning i RWD-läge till en altitud på cirka 50 meter, där en lyckad övergång till FWD-läge genomfördes vid cirka 15km/h. Under övergången sänktes drönarens altitud med cirka 7.2 meter. Testet fortsatte med att flyga drönaren i cirklar med cirka 20 meter radie med kontrollerad minskning av altitud. Under flygningen nådde drönaren en högsta horisontell hastighet av 69,9km/h. När drönarens altitud nådde 15 meter över mark, vid en hastighet av 49 km/h, utlöstes en övergång till RWD-läge samt autonom navigering till dess startpunkt. Anledning till detta är en inställning på drönaren som skulle förhindra FWD-flygning vid för låg altitud av säkerhetsskäl men var inte avsiktligt vid detta test. På grund av övergången blev drönaren instabil och operatören förlorade kontrollen över drönaren som därmed kraschade. Av okänd anledning samlade pixhawken inte heller in data efter att drönaren utlöste övergången tillbaka till RWD-läge.

7.3 Drönarens egenskaper och prestanda utifrån flygtesterna

Den slutgiltiga vikten på drönaren efter färdig konstruktion är cirka 2,75 kg, varav cirka 1,45 kg består av den inköpta flygplansdrönaren med batteriet och resterande från tillbyggnaden tillsammans med övriga komponenter. Drönaren är kapabel till att flyga i RWD-läge och kräver åtminstone 70% effekt från vertikala motorer för att hålla sig luftbunden. Den längsta antecknade flygtiden i RWD-läge var cirka 6 minuter och 30 sekunder. Drönaren visades vara svårstyrd när batteriets laddningsnivå blir låg, under 300 mAh. Högsta horisontella hastighet i RWD-läge är 39,4 km/h med flygning med höjdreglering och 26,6 km/h med positionsreglering aktiverad. Högsta antecknade vertikala hastigheten i RWD-läge var 11,6 km/h. I FWD-läge antecknades den högsta horisontella hastigheten till 69.9 km/h. Kryssningshastigheten visades vara ungefär 50km/h och för att övergå från RWD-läge till FWD-läge

krävdes en hastighet på cirka 40 km/h.

7.4 Jämförelse av drönarens effektivitet i RWD-läge och FWD-läge

Data från pixhawken angående den genomsnittliga strömförbrukning i vardera flygläge analyserades. De som användes vid beräkningen nedan var från två av de längsta flygningarna i vardera läge för att få ett bättre genomsnitt. Genomsnittet för RWD-läge var 25,55 A och för FWD-läge 7,5 A. Med 300 mAh som minsta funktionella laddningsnivå beräknades den maximala flygtiden med ett fulladdat batteri till 7,05 minuter för RWD-läge enligt ekvation 7.1 nedan och 24 minuter för FWD-läge enligt ekvation 7.2 nedan.

$$t_{RWD} = \frac{3000mAh \cdot 60min}{25,55A} \tag{7.1}$$

$$t_{FWD} = \frac{3000mAh \cdot 60min}{7,5A} \tag{7.2}$$

Datan om drönarens flygdistans, batterispänning och förbrukade batteriladdning används till att beräkna drönarens energiförbrukning per meter. Endast de delar där hastigheten var någorlunda konstant används för att få fram statistiken för att minska påverkan från hastighetsförändringar. För flygning i RWD-läge fanns ett intervall på 10 sekunder under autonom flygning med konstant höjd och försumbar hastighetsförändring. Datan under denna period samlades och sammanställdes för att beräkna energiförbrukningen per meter till 94,38 J/m enligt ekvation 7.3 nedan som i sin tur användes för att beräkna den teoretiska flyglängden till 1694 m enligt ekvation 7.4 nedan. För FWD-läge används data som samlades under ett intervall på 42 sekunder då drönaren cirkulerade med en genomsnittlig nedstigningshastighet på ungefär 1 m/s. På grund av nedstigningen adderas en marginal jämförbart med den förändrade lägesenergin under tidsperioden, 1133 J. Detta ledde till en energiförbrukning per meter på 9,14 J/m enligt ekvation 7.5 och teoretisk flyglängd på 17 490 m genom ekvation 7.6. Båda flyglängder beräknades med 300 mAh marginal för funktionell flygning.

$$Energiförbrukning\ per\ meter_{RWD} = \frac{70mAh \cdot 14,25V \cdot 3600s}{38,05m} \tag{7.3}$$

$$Flyglängd_{RWD} = \frac{3000mAh \cdot 14,8V \cdot 3600s}{94,38J/m} \tag{7.4}$$

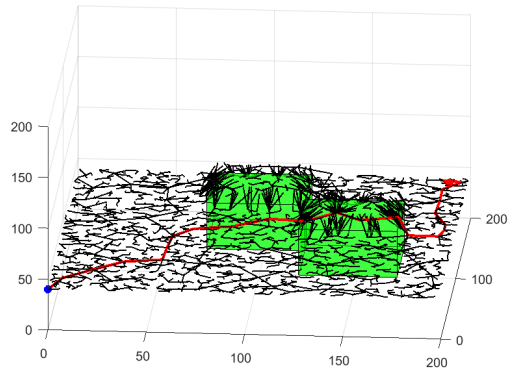
$$Energiförbrukning\ per\ meter_{FWD} = \frac{60mAh \cdot 16V \cdot 3600s + 1133J}{502,13m} \tag{7.5}$$

$$Flyglängd_{FWD} = \frac{3000mAh \cdot 14,8V \cdot 3600s}{9,14/m} \tag{7.6}$$

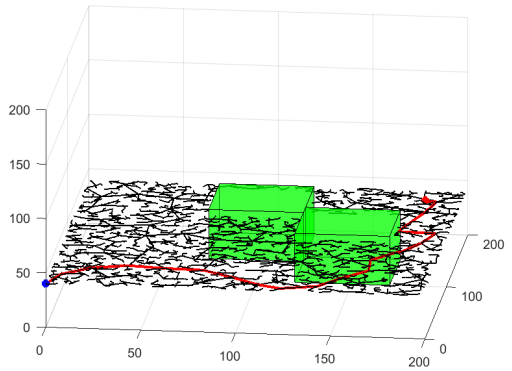
Teoretiskt beräknat har FWD-läget 240% längre flygtid än RWD-läget och 930% längre flyglängd. Ytterligare test, där drönaren startar i RWD-läge, övergår till FWD-läge för att flyga maximal längd och sedan övergår tillbaks till RWD-läge, krävs för ett mer rättvist resultat av hybriddrönaren. Det kan även ses att den teoretiska flygtiden och flyglängden är bättre än den uppmätta, vilket tyder på att flygtestet inte var optimalt.

7.5 Simulering av ruttplaneringsalgoritmer i Matlab

Nedan presenteras resultaten från MatLab simuleringarna av ruttplaneringsalgoritmen, se kapitel 5.2 för genomförande. I figur 7.2 nedan har en simulering plottats i en miljö med två låga hinder. Trädet som byggs upp av besökta noder börjar i simuleringarna i en blå startpunkt som är förutbestämd och avslutas då det når den röda slutpunkten. De svarta linjerna visar hur besökta noder är ihopkopplade och representerar hur trädet brer ut sig. Den röda linjen är vägen från startpunkt till målpunkt som slutligen hittas. I figur 7.2a syns simuleringen av RRT$_{lyft}$. Det går att se att trädet sträcker ut sig i ett plan för att sedan vid de gröna hindrena sträcka sig upp en bit i z-led och röra sig över hindrena för att sedan återgå till samma plan. I 7.2b syns RRT$_{plan}$ i samma miljö. Där har trädet inte lämnat planet utan rört sig runt hindrena.



(a) Bilden visar hur RRT-trädet breder ut sig och hittar en väg över hindrena med RRT$_{lyft}$-algoritmen.



(b) Bilden visar hur RRT-trädet breder ut sig och hittar en väg runt hindrena med RRT$_{plan}$-algoritmen.

Figur 7.2: RRT-simulering i MatLab med två låga hinder i en miljö på 200^3 volymenheter. De gröna boxarna i figuren är hinder och det svarta nätet visar RRT trädet bestående av besökta noder. Den röda linjen är den ihopsammanhängande vägen som algoritmen har hittat mellan startpunkt (blå) och målpunkt (röd).

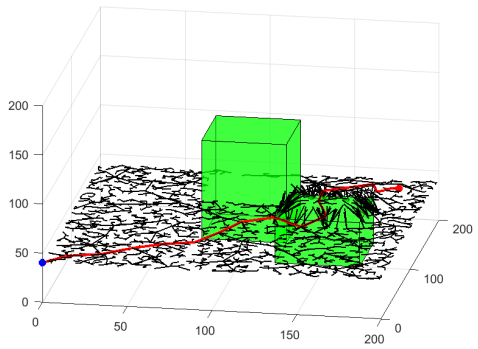
Plottarna används för att visuellt se hur träden breder ut sig och kontrollera att koden gör det som förväntas. Det genomfördes även en mängd tester på både RRT_{lyft} och RRT_{plan} för att jämföra skillnader i sträcka på den väg som de olika algoritmerna hittar, se fullständig tabell i bilaga E. I tabell 7.1 nedan redovisas genomsnitt, minimi- och maximivärden för längden på vägen som hittas för respektive algoritm i miljön sedd i figur 7.2 ovan. Simuleringarna gjordes 25 gånger för respektive algoritm för att få fram ett genomsnitt. Då RRT-algoritmens träd rör sig slumpmässigt blir resultaten olika varje gång.

Tabell 7.1: Två låga hinder

	RRT_{lyft}	**RRT_{plan}**
Test	Sträcka [l.e.]	Sträcka [l.e.]
Max	402,972	501,932
Min	298,752	322,896
Genomsnitt	341,176	372,696

Vid två låga hinder blev genomsnittssträckan för RRT_{lyft} 341,176 längdenheter, medan för RRT_{plan} blev den 372,696 längdenheter. RRT_{lyft} gav därmed i genomsnitt en sträcka som var cirka 8,5 procent kortare än RRT_{plan}. Den genomsnittliga beräkningstiden för RRT_{lyft} är 29.466 s och för RRT_{plan} på 24,942 s. Detta är väldigt långa beräkningstider som troligvis beror på att tidskomplexiteten för koden är $\mathcal{O}(n^2)$. Detta är en väldigt långsam och ineffektiv kod då det inte fanns tid att effektivisera den.

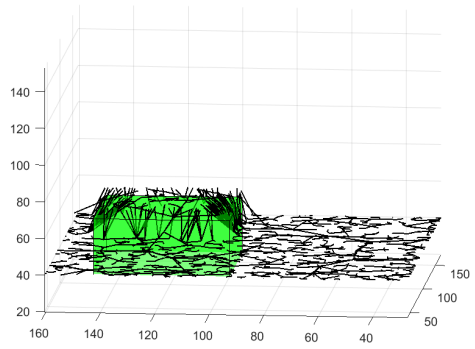
Simuleringar gjordes även för en miljö med ett lågt hinder och ett högt. Plotten för detta syns i figur 7.3 nedan. För denna miljö gjordes inga tester, utan plotten visar endast att trädet rör sig runt det höga hindret, men över det låga, i enlighet med algoritmen. Detta då det höga hindret sträcker sig för högt över planet som trädet rör sig i.



Figur 7.3: Simulering av RRT-trädet i en miljö med ett högt och ett lågt hinder.

7.5.1 Problematik med MatLab-koden och simuleringar

En del problem uppstod med simuleringarna som gav oförväntade eller oönskade resultat. I figur 7.4 nedan syns att trädet skär igenom kanten på ett hinder. Detta sker när hindret är tillräckligt lågt för att den slumpade punkten ska ligga ovanför hindret, men punkten som trädet rör sig från ligger såpass lågt och nära att trädet skär igenom kanten. Detta beror på att en fullständig funktion för kollisionsundvikning i tre dimensioner inte hunnit göras.



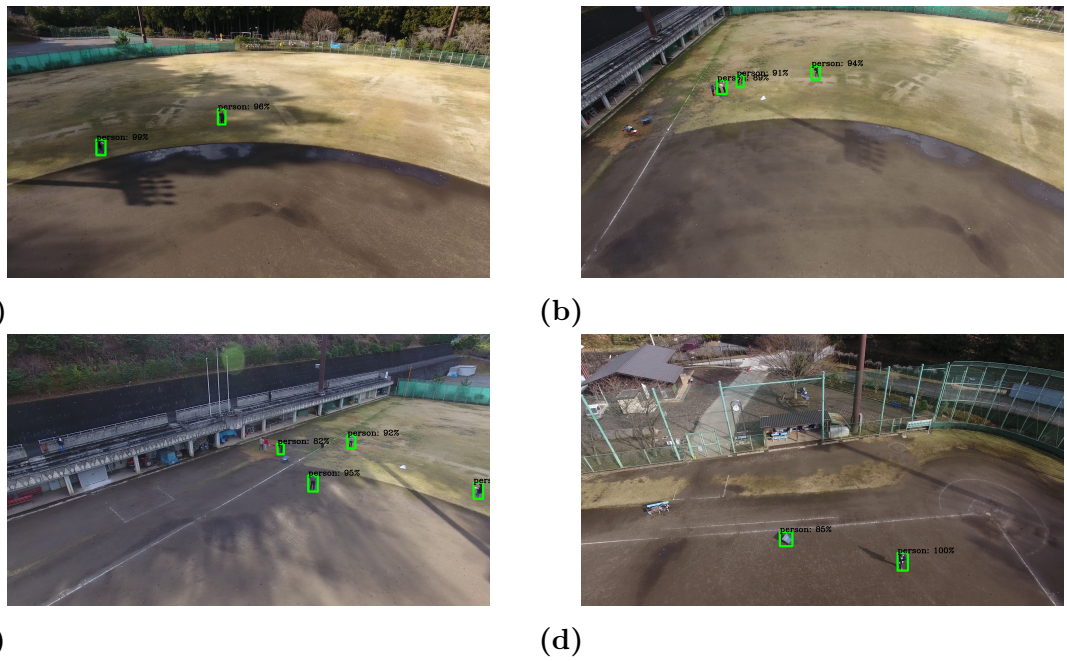
Figur 7.4: Simulering där RRT-trädet skär genom kanten på ett hinder

För enstaka simuleringar av algoritmen noterades också att vägen som RRT_{lyft} hittade gick runt hindren. Detta beror med stor sannolikhet på att valet av punkter i algoritmen sker slumpvis och den första vägen mellan startpunkt och målpunkt som erhålls blir den valda vägen. Ytterligare ett uppmärksammat fall var då RRT_{lyft} hittade en väg över hindrena vars sträcka var större än genomsnittet av sträckan för vägen som RRT_{plan} beräknar. Detta beror på att RRT_{lyft} kan bli ineffektiv i ytterligare ett led och gå mycket upp och ned i onödan. Genomsnittssträckan för RRT_{lyft} är dock fortfarande kortare än RRT_{plan}.

7.6 Resultat av anpassad objektsidentifieringsmodell för detektering av människor

Den färdiga identifieringsmodellens effektivitet testades initialt på videor från datasetet Okutama-Action. Denna testdata hade sparats undan för testerna. Detta då användning av träningsdata vid testning ger ett missvisande resultat, eftersom nätverket tränats på just den datan och då gör bättre ifrån sig. Testvideorna hade dock liknande färg- och ljussättning samt omgivning som träningsdatan. Nedan i figur 7.5 visas några extraherade resultatbilder av detta. Gränsen för när en person beräknas vara upptäckt är då det neurala nätverket anser att det är 80 procent sannolikhet att

en person befinner sig på det aktuella området i bilden. Av testerna framgick det att nätverket har lätt att upptäcka enskilda personer som står upprätt på öppna ytor. Desto större svårigheter hade det med att detektera personer utmed väggar, delvis skymda personer, personer klädda i vissa färger, personer som befinner sig i mörkare ljusförhållanden samt personer i klungor. Ibland klassificerades även felaktiga objekt som människor, vilket figur 7.5d visar. Där klassificerades en skottkärra som människa med 85 procent sannolikhet. Dessutom verkade modellen ofta ha svårt för att upptäcka fler än 5-6 personer åt gången trots att de stod relativt utspridda.



Figur 7.5: Exempel på resulterande bilder där minimumgränsen för klassificering är 80 procents sannolikhet. Från [60]. Omarbetad, CC-BY-NC-SA.

Vid ett senare tillfälle testades identifieringsmodellen även på egenfilmad drönarvideo, där omgivningen var något annorlunda från den som datasetets videor filmats i. Identifieringsmodellen gav då liknande resultat, och det uppskattades att korrekt identifiering skedde främst då videon fångades med cirka 30-70 graders vinkel mot marken, samtidigt som avståndet till de synliga personerna i videon inte var alltför litet eller stort. Den mest avgörande faktorn till huruvida identifieringen blev korrekt var om personerna befann sig på en öppen yta eller inte.

Ytterligare ett Python-script skrevs för att lätt kunna testa och se resultatet från objektsidentifieringsmodellen, även då en bärbar dator med begränsad hårdvara användes. Skriptet sparade en videofil kontinuerligt under tiden som modellen bearbetade indatan. Identifieringsmodellen kunde då testas och resultatet i efterhand spelas upp via videofilen. Detta gav användaren möjlighet att se effektiviteten av modellen i verklig hastighet även om hårdvaran som användes inte uppfyllde de krav som identifieringsmodellen ställer för realtidsidentifiering.

På följande länk kan en resultatvideo nås som visar detta, där videon är filmad med en separat drönare och alltså inte den som utvecklats i projektet:

`http://skandidat.rorgren.se/`

Med en NVIDIA GTX 780 Windforce GPU uppnåde videorenderingen via objektidentiferingsmodellen en hastighet på ca. 31-35 bilder per sekund. Detta är tillräckligt för att visa en videoström från drönaren i realtid, förutsatt att kameran skickar bilddata över nätverket i samma eller lägre hastighet. Med en 2,3 GHz Intel Core i5 CPU uppnåddes i kontrast endast 6-7 bilder per sekund, vilket visar på fördelarna med GPU:er vid användning av CNNs. Teorin bakom detta kan läsas mer om i kapitel 2.4.

8

Diskussion

I detta kapitel diskuteras resultaten presenterade i kapitel 7. Inledningsvis diskuteras hur drönaren och dess förmåga att flyga påverkas av sin vikt. Sedan behandlas resultaten av flygtesterna i övrigt. Vidare diskuteras hur väl ruttplaneringsalgoritmen, utvecklad i detta arbete, fungerar enligt de MatLab simuleringar som presenterats. Slutligen diskuteras hur väl objektidentifieringssystemet fungerar.

8.1 Drönarens manövrerbarhet till följd av ökad vikt

Drönarens vikt vid slutförd konstruktion var 35% tyngre än estimerat. Detta beror på extra komponenter samt delar som inte förutsågs vid estimering och får till följd att de vertikala motorerna kräver mer än 50% effekt för att hålla drönaren luftbunden vilket påverkade drönarens effektivitet i RWD-läge negativt. Trots den ökade vikten anses drönaren vara fullt manövrerbar i RWD-läge. Önskemålet på kryssningshastighet i FWD-läge överstigs med 10km/h, vilket tillsammans med den ökade vikten också kan orsakas av felaktig estimering av lyftkoefficient. Huruvida detta påverkar manövrerbarheten i FWD-läge är svårbedömt utan vidare tester. Drönaren visar däremot markant bättre flygprestanda i FWD-läge relativt till RWD-läge gällande hastighet och energieffektivitet vilket leder till längre teoretisk flygtid och flyglängd. Dock är drönarens prestanda i RWD-läge försämrad av hybriddesignen, vilket betyder att en jämförelse med en annan rotordrönare med samma batteri skulle behövas för en mer representativ jämförelse.

8.2 Utvärdering av flygtester

Utifrån flygtesterna ses att drönaren har säkrast manövrering och är stabilast i RWD-läge med positionsreglering. Däremot har drönaren stundtals svårt att ställa in sig i detta läge vilket leder till en osäkerhet i användningen. Detta tros bero på att

GPS:en har svårigheter att ansluta sig till satelliterna och på så vis stöds inte läget med positionsreglering. Utan denna reglering kan drönaren vara svårstyrd då det hela tiden krävs att manuellt korrigera flygningen. Detta innefattar även läget med höjdreglering vid kraftigare vindar eftersom drönaren har svårt att ge rätt lyftkraft när drönaren inte är parallell med marken.

Vid ruttföljning betedde sig drönaren till en början abnormalt. Den passerade vägpunkten, blev instabil och ändrade rutt. Anledningen till detta är osäkert men vid nästa tillfälle lyckades drönaren följa den förutbestämda rutten bättre med undantag för en kort tidsperiod då drönaren avvek från rutten. En del av flygtesterna utfördes av en oerfaren pilot vilket kan ha påverkat såväl genomförande som resultat. Vid tillfällen upplevdes även kraftigare vindbyar vilka tydligt påverkade drönarens stabilitet i luften. Drönarens data vid dessa tillfällen bör bortses från då förhållandena anses extrema och drönaren föreslås framföras vid lugnare väder tills vidare utveckling skett.

Värdena av regulatorn på flygdatorn är också en faktor som kan har påverkat resultatet, eftersom oscillationer av hög frekvens är ett känt fenomen som inträffar när regulatorns P-verkan är för högt. Orsaken till varför tydliga oscillationer endast uppstod vid flygning med autonom ruttföljning kan bero på att den mänskliga piloten vid manuell flygning försökte hålla drönaren stabil i luften och därmed manuellt reglerat bort de faktorer som orsakar instabilitet för att inte drönaren ska vara till fara för omgivningen. Vidare observerades att drönaren har allvarliga problem att reglera instabilitet samt hålla konstant altitud vid låg batterinivå. Detta tros bero på att batteriet inte klarar av att leverera en stabil spänning till komponenterna.

8.3 Ruttplaneringsalgoritm för hinderundvikning

Från resultatet av simuleringarna framgår att algoritmen presenterad i denna rapport är fullt genomförbar i MatLab simuleringar. Testresultaten påvisar att RRT_{lyft} dessutom ger en kortare väg än RRT_{plan}.

Beräkningar på en rutt som är rimlig för en FWD att flyga har inte gjorts. Skarpa vinklar i rutten som algoritmen hittar kan således vara omöjliga för en FWD att följa i verkligheten. Resultatet visar också att sträckan på rutten som algoritmerna hittar varierar kraftigt på grund av det slumpmässiga beteende RRT-algoritmen grundar sig i. Även om denna algoritm implementeras fullständigt på en drönare går det inte att flyga säkert i en stadsmiljö utan sensorer och dynamisk hinderdetektering. En kombination av statisk och dynamisk kollisionsundvikning hade behövts för att få en drönare att flyga helt autonomt utan risk för att krocka.

Risken att hitta en mindre optimal väg kan öka för RRT_{lyft} om det är många eller stora hinder. Detta då trädet rör sig i ytterligare en dimension och att det då slumpmässigt kan röra sig fram och tillbaka, upp och ned och resultera i en myc-

ket lång och ineffektiv väg. Genomsnittet av de utförda testerna tyder dock på att vägen för RRT_{lyft} ändå blir kortare. Dessa simuleringar är dock inte fullständiga och vidare utveckling samt tester behöver genomföras.

8.4 Objektsidentifieringssystem för detektering av människor

Objektsidentifieringssystemet för detektering av människor visade sig vara väldigt begränsad till den tillgängliga hårdvaran, vilket var något som var förväntat redan i början av utvecklingsprocessen. En förutsättning för att den lösning som utvecklats skall kunna fungera i realtid är därför att markstationen har en tillräckligt kraftfull GPU, vilket var ett antagande som gjordes tidigt redan under avgränsningar i projektet.

Identiferinssmodellen upptäcker och klassificerar ibland andra objekt än personer som människor, vilket figur 7.5d i kapitel 7.6 visar. Detta händer främst då dessa objekt befinner sig på stora platta och öppna ytor, likt hur de flesta instanser av personer i datasetet befann sig på ett öppet fält. Det verkar därför som att nätverket lärt sig att identifiera objekt som befinner sig på öppna ytor snarare än just människor. För att motverka detta hade antagligen mer varierande bilddata behövts användas till träning av nätverket, där även icke-mänskliga objekt placerats i bild tillsammans med människor i avsikt att nätverket bättre skulle kunna lära sig att särskilja dessa. Genom ett tillräckligt stort och varierat dataset hade därav resultatet antagligen kunnat förbättrats mycket.

Som nämns i kapitel 1.3 avgränsade sig projektet till att undersöka huruvida en vanlig bildström skulle vara användbar till att detektera personer. Detta trots att tidigare liknande tillämpningar av maskininlärning för att detektera personer använder sig av värmekamera med lyckat resultat. En stor anledning till detta var en förhoppning om att den erhållna videoströmmen även skulle kunna användas till vidareutvecklingar av objektsidentifieringssystemet för exempelvis hinderundvikning och identifiering av landningsplatser. Dock visar det erhållna resultatet i kombination med resultat från tidigare rapporter så som P.Rudol och P.Doherty's rapport "Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery" [6] att det förmodligen är bättre att använda just värmekamera för specifikt persondetektering. Detta då identifieringsmodellen inte blir lika beroende av att sikten är klar, samtidigt som systemet blir mer robust för olika typer av vinklar om det kan tränas att känna igen värmekällor i bilden istället för former av människor. Dessutom bör separata bildströmmar implementeras för detektering av människor respektive andra typer av objekt. Dock tillåter den utvecklade lösningen med YOLO och Darkflow, vilka kan läsas mer om i kapitel 2.4.3, att en modell enkelt kan anpassas för användning med en värmekamera istället. Det som då krävs är ett nytt dataset med denna typ av bilddata.

Som figur A.1 i appendix A visar så är förlusten vid träningen relativt konstant redan efter 10 000 steg. Detta innebär i detta fall att det inte görs några märkbara förbättringar när det kommer till hur väl det neurala nätverket anpassat sig till träningsdatan efter denna punkt. Detta motsvarar en träningstid på ca. 2,5 - 3 timmar. Den största anledningen till att träningen kunde genomföras så pass snabbt är datasetets begränsade storlek och varians, vilket gör att det neurala nätverket snabbare anpassar sig rätt efter träningsdatan.

Resultatet visade även på att avstånd och kameravinkel spelar en avgörande roll i hur korrekt identifieringen av människor blir. Även här hade ett större dataset med fler varierande bilder kunnat skapa ett mer robust system. Dock är kombinationen av dessa två parametrar ett mer komplext problem då införandet av bilddata som är helt olik den resterande bilddatan i datasetet kan påverka resultatet ofördelaktigt, genom bieffekter så som att modellen lättare klassificerar felaktiga objekt som människor. Om ett mycket precist eller noggrant resultat önskas uppnås bör istället olika modeller utvecklas för olika situationer, det vill säga ifall drönaren skall flygas på hög eller låg höjd; om en brant eller flack vinkel mot marken önskas användas och om drönaren bör kunna upptäcka personer på nära eller långt avstånd.

8.5 Samhälleliga och etiska aspekter

På senare tid har drönare blivit allt mer populära både som professionellt hjälpmedel och som hobby för privatpersoner. Ökningen av antalet drönare och utvecklingen av dessa har lett till en omfattande diskussion om säkerhet och integritet [63]. Detta har i sin tur resulterat i nya lagar kring användandet av drönare [7]. Om något fel uppstår i mjukvara eller hårdvara finns en klar risk för att drönaren flyger in i något eller någon och orsakar skador på egendom eller person. Även handhavandefel kan resultera i allvarliga konsekvenser. Autonomi utgör en komplex diskussion om ansvarstagande och skuldbeläggning där det är svårt att dra gränser. Vid en olycka kan det bli problematiskt att avgöra om det är tillverkaren eller användaren av drönaren som skall hållas ansvarig. En drönare som dessutom är utrustad med kamera kan användas för övervakning av personers privatliv och inskränker på så vis personers integritet.

Den färdiga produkten skulle dock kunna innebära en fantastisk resurs för till exempel brandkåren. Släckning- och räddningsinsatser kan bli säkrare och mer information om branden skulle finnas tillgänglig, exempelvis omfattning och risk för personskada. Drönare har många tekniska fördelar och ger helt nya möjligheter att övervaka, transportera och undersöka utan att människor behöver befinna sig fysiskt nära. En liten drönare kan dessutom ta sig fram och komma åt i små, trånga utrymmen som människor inte kan nå. Det kan bespara resurser och människors säkerhet i stor utsträckning. När drönaren används i dess avsedda syfte, exempelvis

att söka efter och larma om bränder eller transportera medicin, skulle drönaren ha en positiv påverkan på de personer vars liv eller egendom kan räddas med hjälp av drönaren. Så fort den flygs bland allmänheten kan dock ofrivillig övervakning bli en konsekvens och ljudet från drönare kan bli ett störningsmoment för både djur och människor [64].

I genomförandet av detta projekt anses inskränkning av individers integritet samt säkerhet de huvudsakliga problemen. För att undvika detta utfördes flygtester av drönaren på ett avgränsat område. Precis som med all teknik går drönare även att använda med avsikter som kan anses omoraliska eller problematiska. Att en drönare kan ta sig fram där människor inte kan, kan naturligtvis resultera i att känslig information sprids. Med viss utrustning går en drönare även att göra om till ett vapen eller använda i kriminella syften [65]. Dilemmat är att drönare har många olika användningsområden och går att använda lika väl som en resurs vid räddningsinsatser som tveksam övervakning eller krigföring [65].

Utfallet av detta projekt kan därav bli etiskt problematiskt. Ett sätt att ge individer möjlighet att välja om de vill utsätta sig för drönarövervakning är att informera om de områden som systematiskt övervakas med drönare. I övrigt då drönare flygs in vid akuta behov kan det anses vara av högre prioritet att bistå i en räddningsinsats än att någons integritet inskränks [66]. Det blir en avvägning som får göras för varje unik situation för att se om fördelarna vid användning av drönare väger upp för nackdelarna.

9

Slutsats

Projektet syftade till att undersöka, dokumentera och utveckla ett högautomatiserat system för användning i sök- och räddningsuppdrag. Detta system skulle grunda sig i konstruktion av en hybrid mellan en rotordrönare och en flygplansdrönare med ett objektsidentifieringssystem. En slutprodukt skulle enligt målet ha autonom förmåga att starta och landa vertikalt, hovra samt flyga långa distanser. Persondetekteringen skulle implementeras med hjälp av maskininlärningsalgoritmer.

Genomförandet av arbetet utfördes i fyra separata delar med avsikten att koppla ihop detta i ett senare skede. Dessa delar innefattade: konstruktion och konfiguration av hybriddrönare samt trådlös överföring av information, mätdata och realtidsström, den autonoma flygningen för drönaren och objektsidentifiering för detektering av människor.

Projektet lyckades aldrig sammanställa de olika delarna i en första prototyp av det sammankopplade sök- och räddningssystemet. Detta då det krävs att en lösning av realtidsvideoströmningen från drönaren utvecklats innan det kompletta systemet kan anses uppnå ett första stadie. Samtidigt är det önskvärt att uppnå en mer avancerad nivå av autonom flygning än den som implementerats med hjälp av flygdatorn Pixhawk 4. Ruttplaneringsalgoritmen som utvecklats i detta syfte kräver både effektivisering, vidareutveckling och implementering för att användas. Även dynamisk kollisionsundvikning bör finnas i en slutprodukt för att säkerställa att drönaren ej kolliderar med hinder. Persondetekteringsmodellen hade kunnat användas i en första systemprototyp, men i en slutprodukt krävs anpassning för användning med eller i kombination med värmekamerabilder, trots att den nuvarande implementationen med ett dataset av större varians hade kunnat förbättra resultatet avsevärt.

Detta arbete lyckas ej sammankoppla ett fullständigt system, men flera delmål uppnåddes som visar på lovande utvecklingspotential. Drönaren har möjlighet att flyga i både RWD-läge och FWD-läge, där resultaten tyder på att den kan flyga långa distanser samtidigt som den behåller de önskvärda egenskaper rotordrönare erbjuder. Det är även troligt att systemet som projektet syftat i att utveckla skulle kunna vara till stor nytta bland fler områden än sök- och räddning, då de olika tekniker som används kan vidareutvecklas för många olika typer av tillämpningar.

Litteratur

[1] "Military Aircraft," i *Britannica Academic*. [Online]. Tillgänglig: https://academic.eb.com/levels/collegiate, hämtad: 2019-02-11.

[2] F. Nex och F. Remondino, "UAV for 3D mapping applications: a review", *Applied Geomatics*, årg. 6, nr 1, s. 1–15, mars 2014, Hämtad: 2019-02-26, ISSN: 1866-928X. DOI: 10.1007/s12518-013-0120-x. URL: https://doi.org/10.1007/s12518-013-0120-x.

[3] T. Dietrich, S. Krug och A. Zimmermann, "An empirical study on generic multicopter energy consumption profiles", i *2017 Annual IEEE International Systems Conference (SysCon)*, april 2017, s. 1–6. DOI: 10.1109/SYSCON.2017.7934762.

[4] D. Floreano och R. J. Wood, "Science, technology and the future of small autonomous drones", *Nature*, årg. 521, s. 460, 2015, Hämtad: 2019-02-11. DOI: 10.1038/nature14542. URL: https://doi.org/10.1038/nature14542.

[5] J. Redmon, S. Divvala, R. Girshick och A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", i *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, juni 2016.

[6] P. Rudol och P. Doherty, "Human Body Detection and Geolocalization for UAV Search and Rescue Missions Using Color and Thermal Imagery", i *2008 IEEE Aerospace Conference*, mars 2008, s. 1–8. DOI: 10.1109/AERO.2008.4526559.

[7] "Transportstyrelsens författningssamling: Transportstyrelsens föreskrifter om obemannade luftfartyg," Transportstyrelsen, Norrköping, Sverige, TSFS 2017:110, 2017. [Online]. Tillgänglig: https://www.transportstyrelsen.se/TSFS/TSFS%202017$_1$10.*pdf*, *hmtad* : 2019−05 − 16..

[8] M. Munk, "The Aerodynamic Forces on Airship Hulls", NASA, tekn. rapport, 1979. URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19930091249.pdf.

[9] Auawise, "The position of all three axes, with the right-hand rule for its rotations," 2010. [Elektronisk bild]. Tillgänglig: http://bit.do/eStJM, hämtad: 2019-05-15.

[10] National Aeronautics and Space Administration, “What is Lift?,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/lift1.html, hämtad: 2019-05-02.

[11] National Aeronautics and Space Administration, “The Lift Equation,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/lifteq.html, hämtad: 2019-05-02.

[12] National Aeronautics and Space Administration, “The Lift Coefficient,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/liftco.html, hämtad: 2019-05-02.

[13] National Aeronautics and Space Administration, “Inclination Effects on Lift,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/incline.html, hämtad: 2019-05-02.

[14] M. S. Selig, *Summary of low speed airfoil data*. SoarTech, 1995, vol. 1.

[15] National Aeronautics and Space Administration, “Aircraft Rotations,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/rotations.html, hämtad: 2019-05-02.

[16] National Aeronautics and Space Administration, “Cruise - Balances Forces,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/cruise.html, hämtad: 2019-05-09.

[17] National Aeronautics and Space Administration, “The Drag Equation,” 2019. [Online]. Tillgänglig: https://www.grc.nasa.gov/www/k-12/airplane/drageq.html, hämtad: 2019-05-09.

[18] W. Johnson, *Helicopter Theory*. Dover Publications, 1980, [Online]. Tillgänglig: https://app.knovel.com/hotlink/toc/id:kpHT000012/helicopter-theory/helicopter-theory, hämtad: 2019-05-02., ISBN: 978-0-486-68230-3.

[19] R. Ward, “The Long Road to the Tiltrotor,” 2018. [Online]. Tillgänglig: https://www.ainonline.com/aviation-news/business-aviation/2018-04-06/long-road-tiltrotor, hämtad: 2019-05-09.

[20] Juliandanzer, “Printable Tiltwing Tiltrotor Drone Body 3D print model,” 2018. [Elektronisk bild]. Tillgänglig: https://www.cgtrader.com/3d-print-models/hobby-diy/robotics/printable-tiltwingtiltrotor-drone-body, hämtad: 2019-05-14.

[21] R. H. Stone, P. Anderson, C. Hutchison, A. Tsai, P. Gibbens och K. Wong, ”Flight testing of the T-wing tail-sitter unmanned air vehicle”, *Journal of Aircraft*, årg. 45, nr 2, s. 673–685, 2008.

[22] USAF, “SkyTote,” 2004. [Elektronisk bild]. Tillgänglig: http://bit.do/eStED, hämtad: 2019-05-15.

[23] G. Flores och R. Lozano, ”Transition flight control of the quad-tilting rotor convertible MAV”, i *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, maj 2013, s. 789–794. DOI: 10.1109/ICUAS.2013.6564761.

[24] canberrauav, “Telemaster Quadplane”, 2015. [Elektronisk bild]. Tillgänglig: http://bit.do/eTFhB, hämtad: 2019-05-15.

[25] Open systems interconnection model (OSI), 35.100.01, 1994.

[26] User Datagram Protocol (UDP), 35.100.01, 1980.

[27] Transmission Control Protocol (TCP), RFC: 793, 1981.

[28] Hypertext Transfer Protocol (HTTP), RFC: 2616, 1999.

[29] Real-Time Streaming Protocol (RTSP), RFC: 7826, 2016.

[30] S. M. Lavalle, ”Rapidly-Exploring Random Trees: A New Tool for Path Planning”, Iowa State University, tekn. rapport, 1998.

[31] A. Hidalgo-Paniagua, J. P. Bandera, M. Ruiz-de-Quintanilla och A. Bandera, ”Quad-RRT: A real-time GPU-based global path planner in large-scale real environments”, *Expert Systems with Applications*, årg. 99, s. 141–154, 2018, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2018.01.035. URL: http://www.sciencedirect.com/science/article/pii/S0957417418300411.

[32] S. Karaman och E. Frazzoli, ”Sampling-based Algorithms for Optimal Motion Planning”, *CoRR*, årg. abs/1105.1186, 2011. arXiv: 1105.1186. URL: http://arxiv.org/abs/1105.1186.

[33] A. K. Guruji, H. Agarwal och D. Parsediya, ”Time-efficient A* Algorithm for Robot Path Planning”, *Procedia Technology*, årg. 23, s. 144–149, 2016, 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05-06 February, 2016, ISSN: 2212-0173. DOI: https://doi.org/10.1016/j.protcy.2016.03.010. URL: http://www.sciencedirect.com/science/article/pii/S2212017316300111.

[34] M. Dakulovi, S. Horvati och I. Petrovi, ”Complete Coverage D* Algorithm for Path Planning of a Floor-Cleaning Mobile Robot”, *IFAC Proceedings Volumes*, årg. 44, nr 1, s. 5950–5955, 2011, 18th IFAC World Congress, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20110828-6-IT-1002.03400. URL: http://www.sciencedirect.com/science/article/pii/S147466701644557X.

[35] P. Lajevardy och A. Mousavian och M. A. Oskoei, A Comparison Between RRT* and A* Algorithms for Motion Planning in Complex Environments", 2nd. International Congress of Electrical Engineering,Computer Science and Information Technology„ 2017 [Online].
Tillgänglig: https://www.researchgate.net/publication/280094486_A_Comparison_Between_RRT_and_A_Algorithms_for_Motion_Planning_in_Complex_Environments, hämtad: 2019-05-31.

[36] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, Tillgänglig: http://planning.cs.uiuc.edu/.

[37] A. Krizhevsky, I. Sutskever och G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", i *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou och K. Q. Weinberger, utg., Curran Associates, Inc., 2012, s. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[38] Y. LeCun, Y. Bengio och G. Hinton, "Deep learning", *nature*, årg. 521, nr 7553, s. 436, 2015.

[39] S. B. Kotsiantis, I. Zaharakis och P. Pintelas, "Supervised machine learning: A review of classification techniques", *Emerging artificial intelligence applications in computer engineering*, årg. 160, s. 3–24, 2007.

[40] Tzutalin, "LabelImg," 2015. [Online]. Tillgänglig: https://github.com/tzutalin/labelImg, hämtad: 2019-04-11.

[41] MTheiler, "Detected with YOLO," 2018. [Elektronisk bild]. Tillgänglig: http://bit.do/eTbEg, hämtad: 2019-05-14.

[42] W. Rawat och Z. Wang, "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review", *Neural Computation*, årg. 29, s. 1–98, juni 2017. DOI: 10.1162/NECO_a_00990.

[43] L. Bottou, "Large-Scale Machine Learning with Stochastic Gradient Descent", i *Proceedings of COMPSTAT'2010*, Y. Lechevallier och G. Saporta, utg., Heidelberg: Physica-Verlag HD, 2010, s. 177–186, ISBN: 978-3-7908-2604-3.

[44] M. Radovic, O. Adarkwa och Q. Wang, "Object Recognition in Aerial Images Using Convolutional Neural Networks", *Journal of Imaging*, årg. 3, nr 2, 2017, ISSN: 2313-433X. DOI: 10.3390/jimaging3020021. URL: http://www.mdpi.com/2313-433X/3/2/21.

[45] Apple Inc., "Core ML," 2019. [Online]. Tillgänglig: https://developer.apple.com/documentation/coreml, hämtad: 2019-04-30.

[46] Apple Inc., "Vision," 2019. [Online]. Tillgänglig: https://developer.apple.com/documentation/vision, hämtad: 2019-04-30.

[47] Thtrieu, "Darkflow," 2018. [Online]. Tillgänglig: https://github.com/thtrieu/darkflow, hämtad: 2019-04-11.

[48] Tensorflow, 2019. [Online]. Tillgänglig: https://www.tensorflow.org/about?hl=sv, hämtad: 2019-05-14.

[49] G. Bradski och A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc., 2008.

[50] J. Redmon och A. Farhadi, "YOLO9000: Better, Faster, Stronger", *CoRR*, årg. abs/1612.08242, 2016. arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[51] O. Liang, *How to choose motor for racing drone & quadcopter*, [Online]. Tillgänglig: https://oscarliang.com/quadcopter-motor-propeller/, hämtad: 2019-05-30, 2017.

[52] Intel Corporation, “MAVlink Router,” 2019. [Online]. Tillgänglig: https://github.com/intel/mavlink-router, hämtad: 2019-05-02.

[53] Telia, *Täckningskartor*. URL: https://www.telia.se/privat/support/tackningskartor.

[54] ——, *Telia Mobilt Bredband*. URL: https://www.telia.se/privat/bredband/abonnemang-kontantkort/produkt/telia-mobilt-bredband.

[55] Silvanmelchior, “RPi-Cam-Web-Interface,” 2019. [Online]. Tillgänglig: https://github.com/silvanmelchior/RPi_Cam_Web_Interface, hämtad: 2019-05-02.

[56] M. Ramana, S. A. Varma och M. Kothari, ”Motion Planning for a Fixed-Wing UAV in Urban Environments”, *IFAC-PapersOnLine*, årg. 49, nr 1, s. 419–424, 2016, 4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.03.090. URL: http://www.sciencedirect.com/science/article/pii/S2405896316300908.

[57] S. Vemprala, *2D/3D RRT**, Hämtad: 2019-04-28, 2017. URL: https://se.mathworks.com/matlabcentral/fileexchange/60993-2d-3d-rrt-algorithm.

[58] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn och A. Zisserman, ”The Pascal Visual Object Classes (VOC) Challenge”, *International Journal of Computer Vision*, årg. 88, nr 2, s. 303–338, juni 2010.

[59] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár och C. L. Zitnick, ”Microsoft COCO: Common Objects in Context”, *CoRR*, årg. abs/1405.0312, 2014. arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[60] M. Barekatain, M. Mart, H. Shih, S. Murray, K. Nakayama, Y. Matsuo och H. Prendinger, ”Okutama-Action: An Aerial View Video Dataset for Concurrent Human Action Detection”, *CoRR*, årg. abs/1706.03038, 2017. arXiv: 1706.03038. URL: http://arxiv.org/abs/1706.03038.

[61] VideoLAN Organization, “VLC Media Player,” 2019. [Online]. Tillgänglig: https://www.videolan.org/vlc, hämtad: 2019-04-10.

[62] J. Redmon, “Darknet: Open Source Neural Networks in c,” 2013–2016. [Online]. Tillgänglig: http://pjreddie.com/darknet, hämtad: 2019-04-11.

[63] E. Holmgren och K. Iliadis, Klassificering av drönare: En systematisk litteraturundersökning, Dissertation, 2016.

[64] A. Christian and R. Cabell, Initial Investigation into the Psychoacoustic Properties of Small Unmanned Aerial System Noise, i 23rd AIAA/CEAS Aeroacoustics Conference, Denver, Colorado, USA, 2017 [Online]. Tillgänglig: https://arc.aiaa.org/doi/10.2514/6.2017-4051, hämtad: 2019-05-16.

[65] L. Johansson, ”Autonomous Systems in Society and War: Philosophical Inquiries”, xavhandling, Filosofi och teknikhistoria, Kungliga Tekniska Högskolan, Stockholm, Sverige, år 2013. [Online]. Tillgänglig: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A646174dswid=-6072, hämtad: 2019-05-16.

[66] Justitiedepartementet: Grundlagsenheten, "Kameraövervakningslagen och möjligheterna att använda drönare för berättigade ändamål," Regeringskansliet, Stockholm, Sverige, 2017. [Online]. Tillgänglig: https://www.regeringen.se/4afefa/contentassets//kameraovervakningslagen-och-mojlighetera-att-anvanda-dronare-for-berattigade-andamal, hämtad: 2019-05-16.

[67] D. Kingma och J. Ba, ”Adam: A Method for Stochastic Optimization”, *International Conference on Learning Representations*, dec. 2014.

[68] M. B. Bueno, X. Giró-i-Nieto, F. Marqués och J. Torres, ”Hierarchical object detection with deep reinforcement learning”, *Deep Learning for Image Processing Applications*, årg. 31, nr 164, s. 3, 2017.

A

Träning av objektsidentifieringsmodell

Detta stycke behandlar träningsprocessen av den objektsidentifieringsmodell som utvecklats mer utförligt. Teorin som behövs för stycket beskrivs i kapitel 2.4.

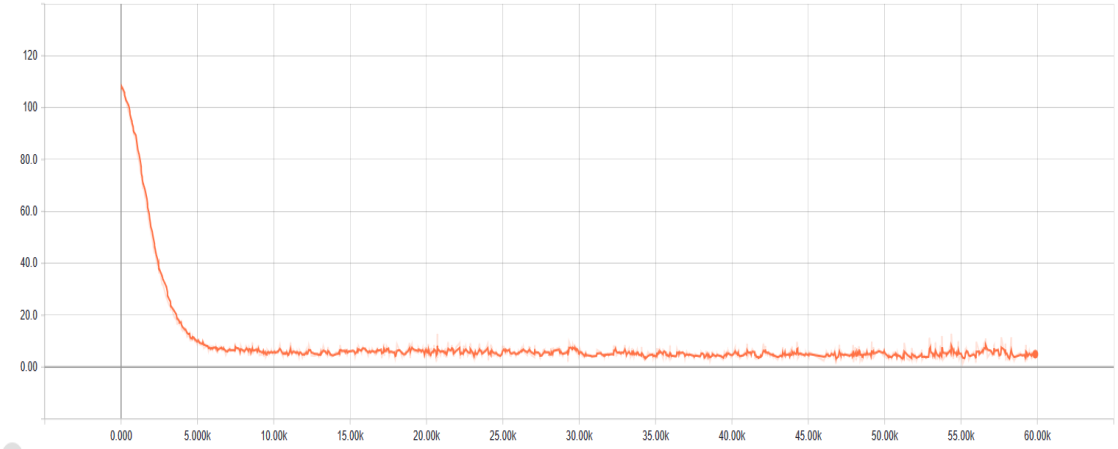
Darkflows träningsprocess erbjuder många möjligheter att anpassas. Bland annat kan learning rate, här kallad för inlärningshastigheten, ändras till godtyckligt värde. Inlärningshastigheten styr hur mycket parametrarna i nätverket påverkas varje steg. Generellt sett ger ett mindre värde en noggrannare modell, medan en högre inlärningshastighet gör att nätverket konvergerar mot en fast kostnad snabbare, och alltså minskar träningstiden. Detta sker dock på bekostnad av en mindre noggrann modell [42].

Träningen utfördes på en NVIDIA GTX 780 Windforce GPU. I den slutgiltiga versionen användes en batch-storlek på 8 bilder, vilket innebär att 8 bilder från datasetet evalueras samtidigt, främst för att processen skall gå snabbare. En inlärningshastighet på 10^{-5} (enhetslös) användes. Denna minskades sedan till 10^{-6} efter ca. 30000 steg, där varje steg motsvarar att en batch har behandlats.

Olika optimeringsalgoritmer går att välja vid träning med Darkflow. Ett exempel är Adam som är en beräkningseffektiv algoritm för stokastisk optimering. Den lämpar sig bra för problem som är stora med avseende på datamängd och antal parametrar, samt då gradienten är brusig [67]. Detta gör den till ett populärt val vid objektsidentifieringstillämpningar [68]. För modellens optimeringsalgoritm valdes därför just denna.

Den anpassade objektsidentifieringsmodellen som tillslut kom att användas tränades under ca. 18 timmar tills dess att sannolikheten ansågs liten för att fler förbättringar skulle erhållas. En batchstorlek på 8 bilder och 59 000 genomförda steg innebär att nätverket uppdaterades 472 000 gånger, och att datasetet på 751 bilder itererades

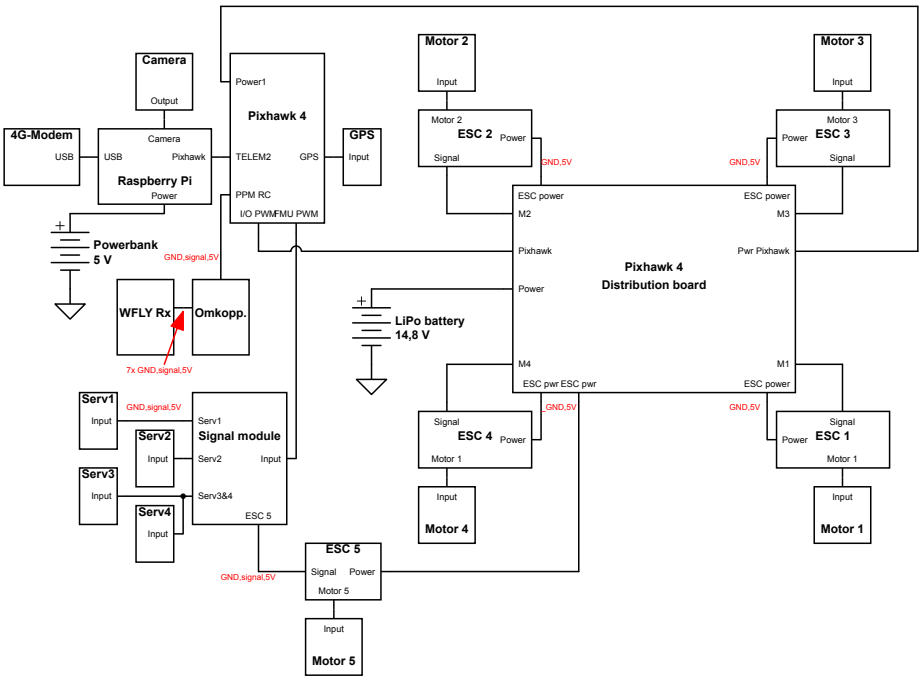
över 628 gånger. I figur A.1 nedan visas den förlust (eng. loss) nätverkets output gav vid varje steg.



Figur A.1: Loss per steg x med batchstorlek 8 och inlärningshastighet 10^{-5} för $0 < x \leq 30000$ och 10^{-6} för $30000 < x \leq 59000$.

B

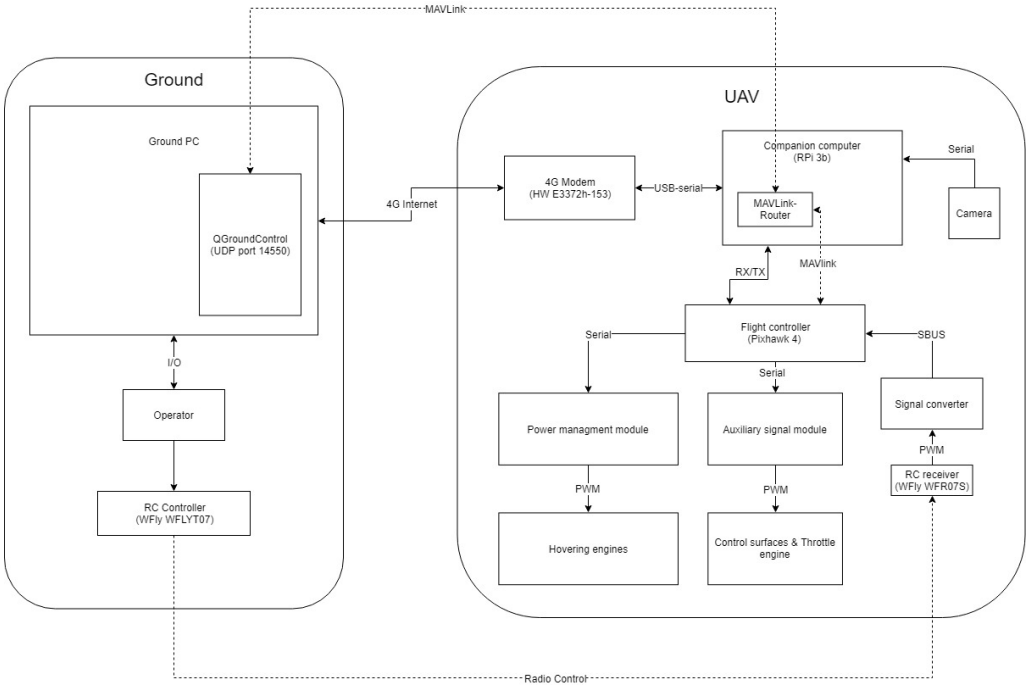
Kretsschema



Figur B.1: Kretsschema för drönarens komponenter

C

Kommunikationsschema

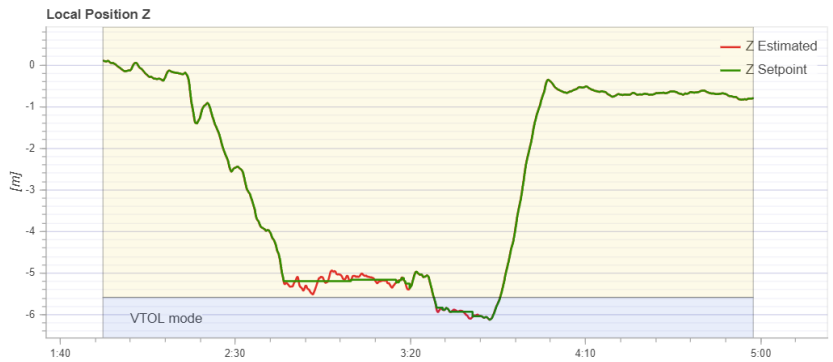


Figur C.1: Detaljerat schema över kommunikationen mellan drönare och markstation

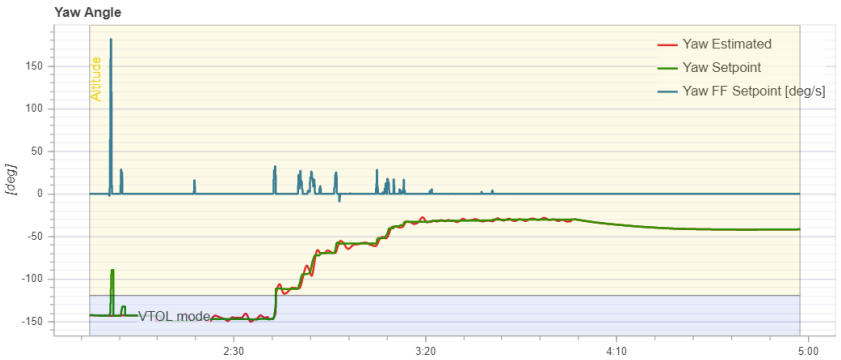
D

Resultat från flygtester

D.1 RWD med altitudreglering



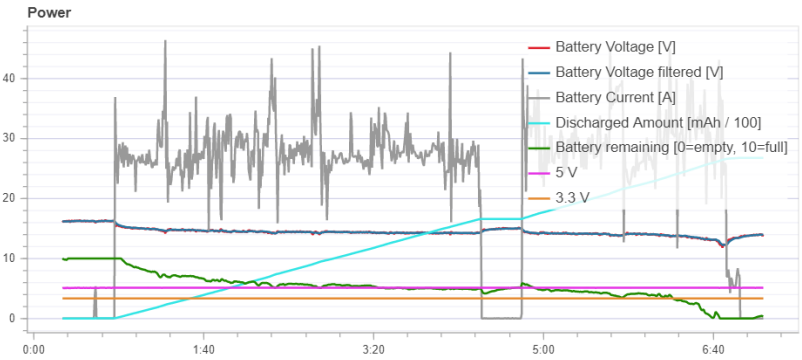
Figur D.1: Jämförelse mellan uppmätt flygdata och börvärde av altitud under altitudtest



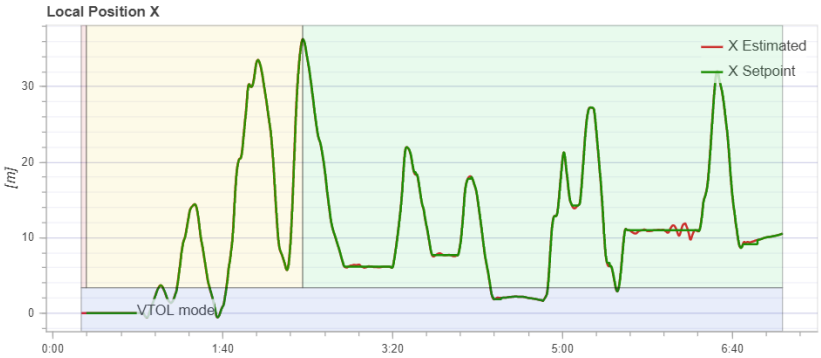
Figur D.2: Jämförelse mellan uppmätt flygdata och börvärde av yaw under altitudtest

Figur D.3: Jämförelse mellan uppmätt flygdata och börvärde av pitch under altitudtest

D.2 RWD med positionsreglering

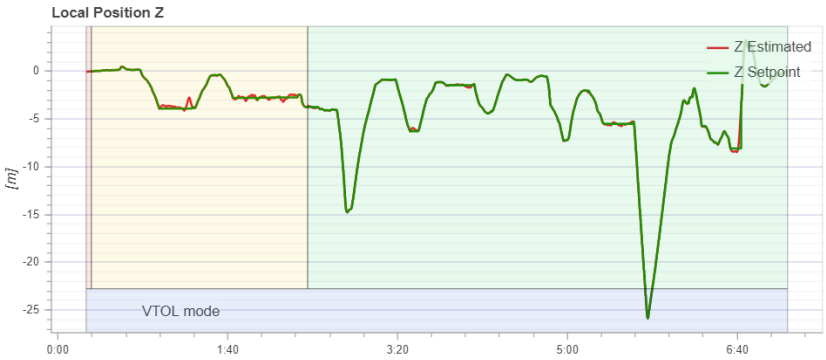


Figur D.4: Batterinivå under positionstest



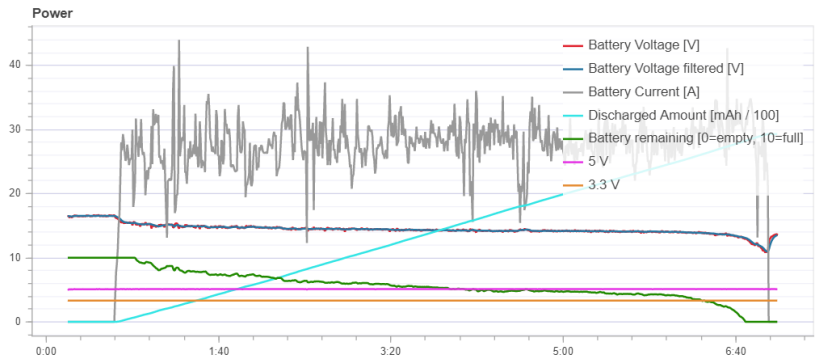
Figur D.5: Jämförelse mellan uppmätt flygdata och börvärde i x-led under positionstest

Figur D.6: Jämförelse mellan uppmätt flygdata och börvärde i y-led under positionstest



Figur D.7: Jämförelse mellan uppmätt flygdata och börvärde i höjdled under positionstest

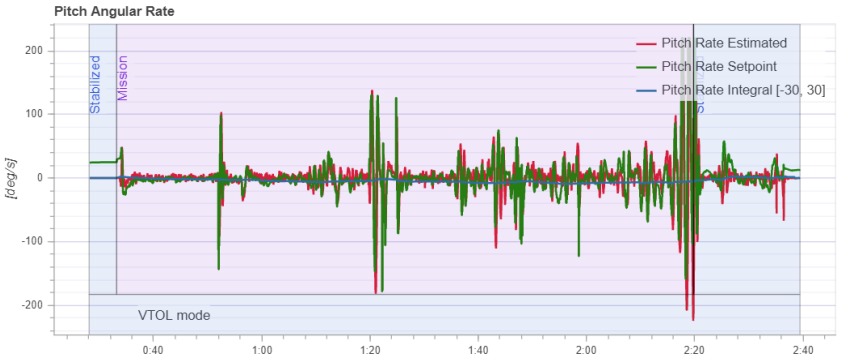
D.3 Flygtid i RWD-läge



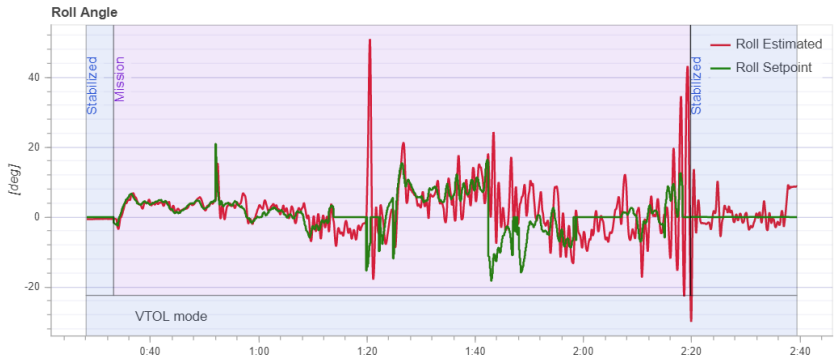
Figur D.8: Batterinivå under uthållighetstest

D.4 RWD med autonom navigering 1

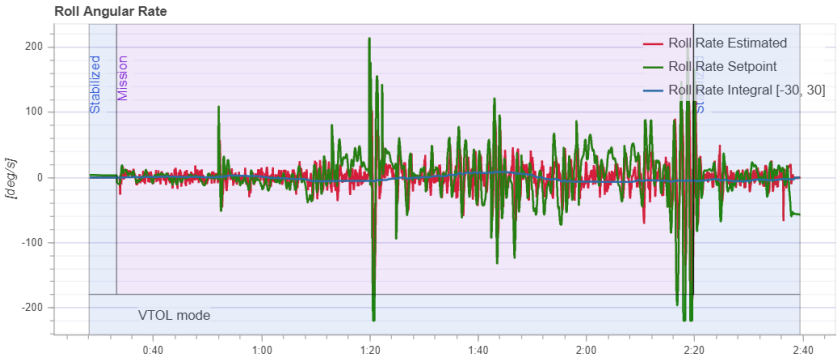
Figur D.9: Jämförelse av pitch mellan uppmätt flygdata och börvärde under ruttföljningstest 1



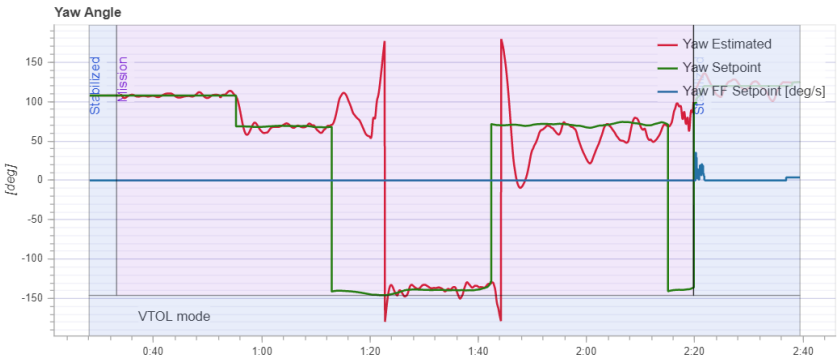
Figur D.10: Jämförelse av pitchhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 1



Figur D.11: Jämförelse av roll mellan uppmätt flygdata och börvärde under ruttföljningstest 1

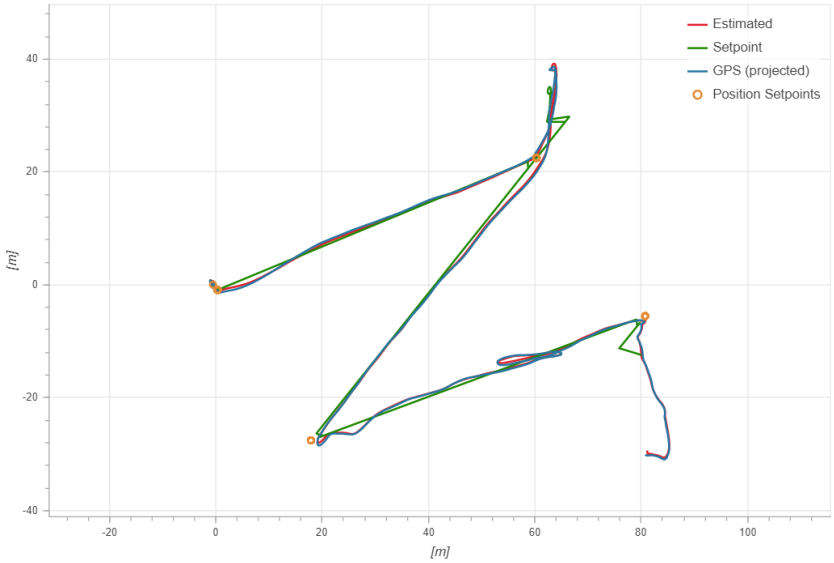


Figur D.12: Jämförelse av rollhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 1



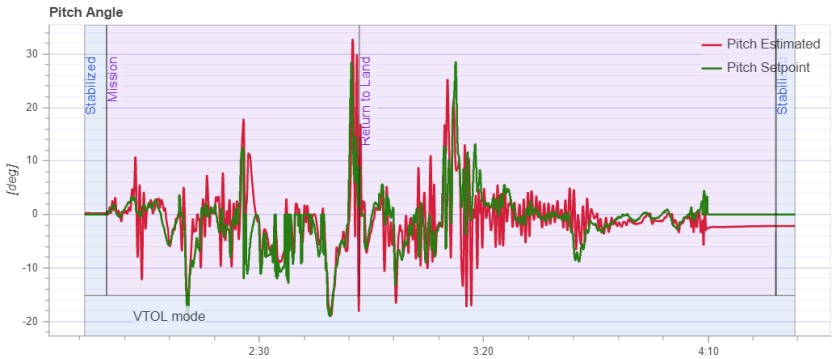
Figur D.13: Jämförelse av yaw mellan uppmätt flygdata och börvärde under ruttföljningstest 1

Figur D.14: Jämförelse av yawhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 1

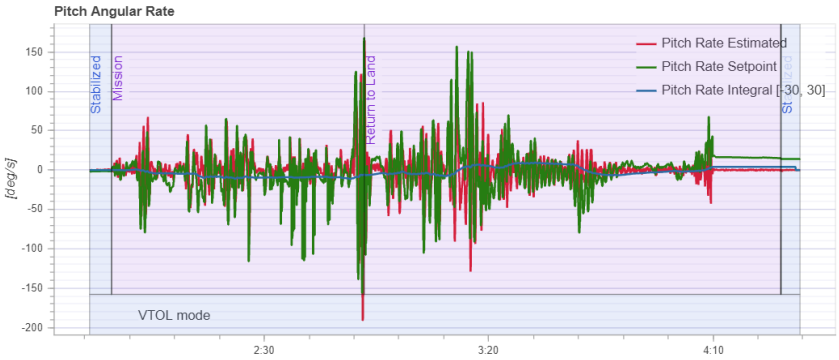


Figur D.15: Jämförelse av rutt mellan uppmätt flygdata och börvärde under ruttföljningstest 1

D.5 RWD med autonom navigering 2

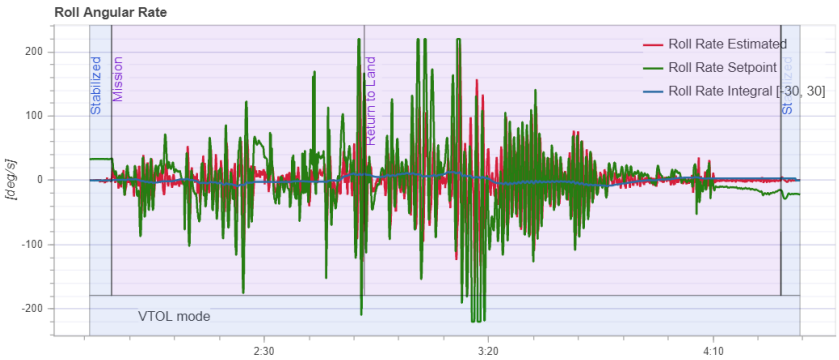


Figur D.16: Jämförelse av pitch mellan uppmätt flygdata och börvärde under ruttföljningstest 2



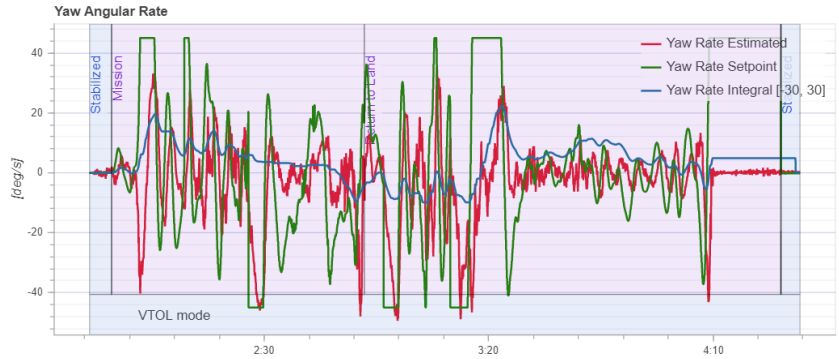
Figur D.17: Jämförelse av pitchhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 2

Figur D.18: Jämförelse av roll mellan uppmätt flygdata och börvärde under ruttföljningstest 2

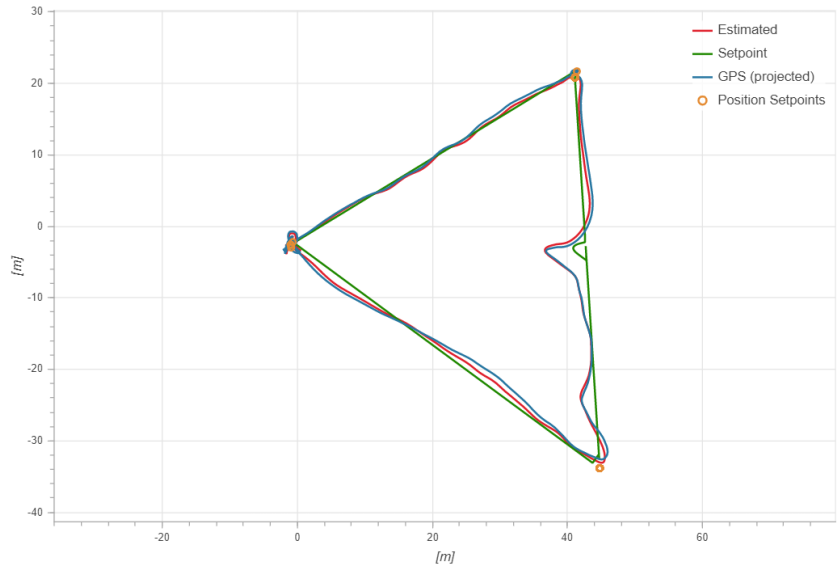


Figur D.19: Jämförelse av rollhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 2

Figur D.20: Jämförelse av yaw mellan uppmätt flygdata och börvärde under ruttföljningstest 2



Figur D.21: Jämförelse av yawhastighet mellan uppmätt flygdata och börvärde under ruttföljningstest 2

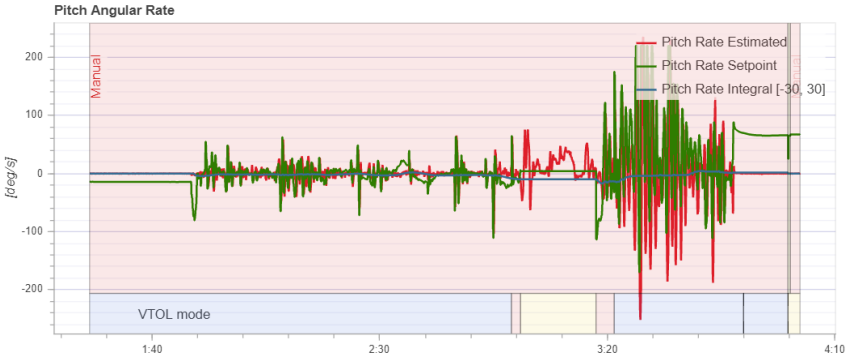


Figur D.22: Jämförelse av rutt mellan uppmätt flygdata och börvärde under ruttföljningstest 2

D.6 Övergång mellan RWD- och FWD-läge

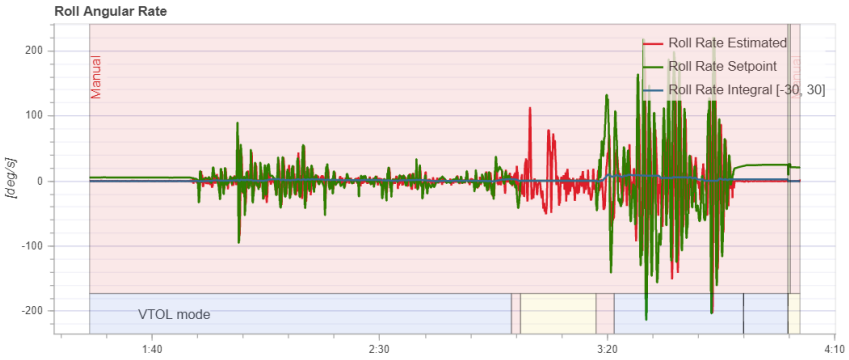
Figur D.23: Höjddata under flygplanstest 1

Figur D.24: Jämförelse av pitch mellan uppmätt flygdata och börvärde under flygplanstest 1



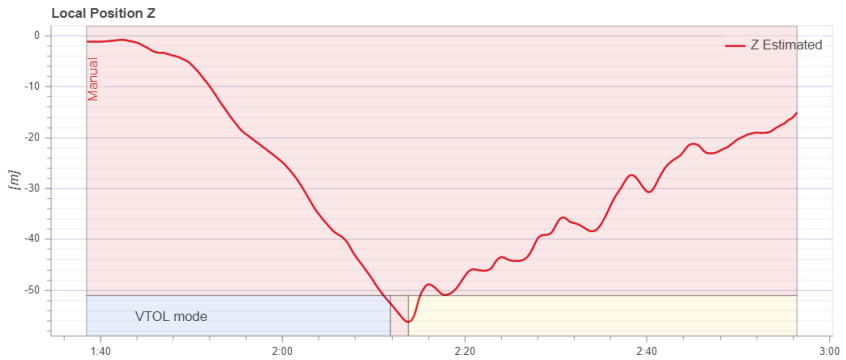
Figur D.25: Jämförelse av pitchhastighet mellan uppmätt flygdata och börvärde under flygplanstest 1

Figur D.26: Jämförelse av roll mellan uppmätt flygdata och börvärde under flygplanstest 1

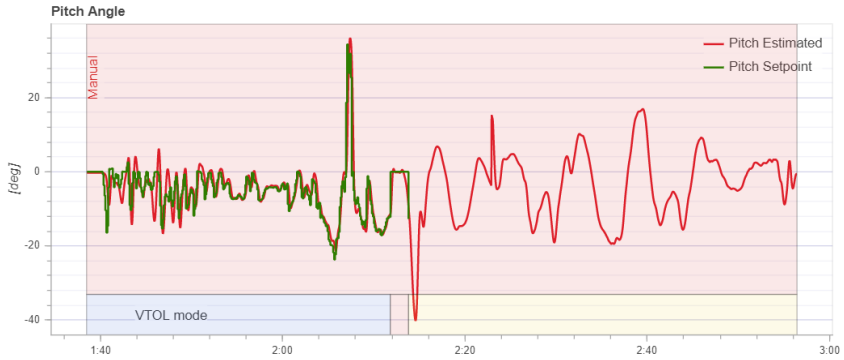


Figur D.27: Jämförelse av rollhastighet mellan uppmätt flygdata och börvärde under flygplanstest 1

D.7 Manövrerbarhet i FWD-läge

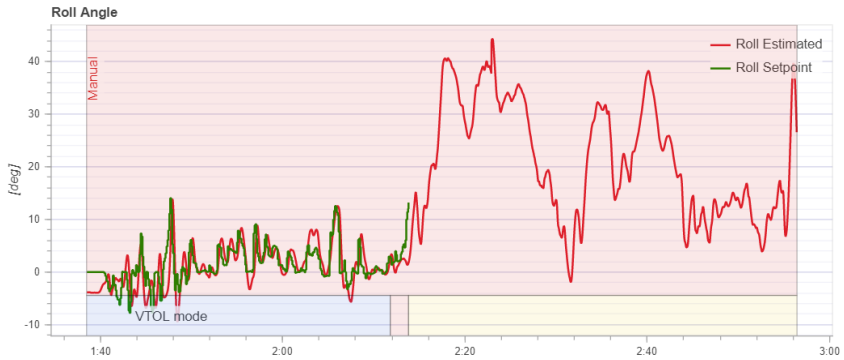


Figur D.28: Höjddata under flygplanstest 2

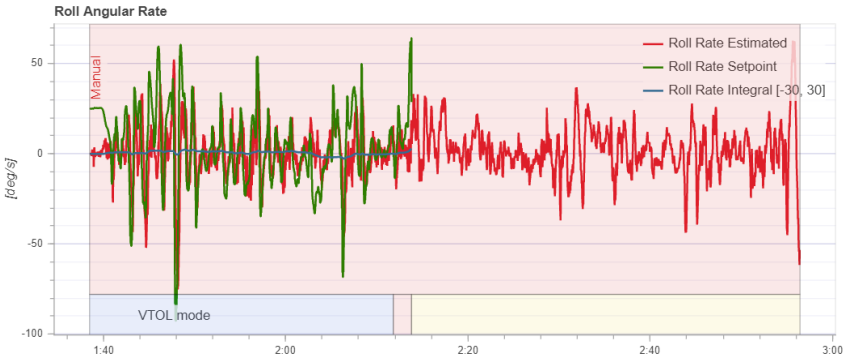


Figur D.29: Jämförelse av pitch mellan uppmätt flygdata och börvärde under flygplanstest 2

Figur D.30: Jämförelse av pitchhastighet mellan uppmätt flygdata och börvärde under flygplanstest 2



Figur D.31: Jämförelse av roll mellan uppmätt flygdata och börvärde under flygplanstest 2



Figur D.32: Jämförelse av rollhastighet mellan uppmätt flygdata och börvärde under flygplanstest 2

E

Tabell simuleringsresultat

	RRT_plan, 2 hinder		*RRT_lyft, 2 hinder*		*RRT_plan, 1 hinder*		*RRT_lyft, 1 hinder*	
Försök	*Avstånd*	*Tid*	*Avstånd*	*Tid*	*Avstånd*	*Tid*	*Avstånd*	*Tid*
1	361.012	26.244	366.111	31.276	399.498	21.300	382.885	31.052
2	371.476	25.658	398.485	30.921	349.213	23.423	319.446	30.689
3	347.742	25.695	309.185	31.238	340.314	20.893	359.144	31.133
4	343.511	26.268	351.609	30.851	385.293	20.982	352.742	30.358
5	441.615	25.871	351.038	33.300	368.878	20.476	304.147	30.685
6	392.669	2.996	321.864	31.001	390.279	20.907	368.619	30.707
7	364.964	25.673	303.485	30.956	334.608	21.237	355.812	30.550
8	340.700	25.294	310.245	0.500	357.225	21.280	339.514	31.136
9	322.896	25.681	383.275	30.888	368.033	20.668	358.024	30.529
10	351.475	25.009	355.083	31.029	409.532	20.830	312.938	30.824
11	373.271	25.460	325.057	31.041	400.187	20.538	369.971	30.480
12	402.535	25.536	353.474	19.416	365.956	21.557	423.488	30.803
13	501.932	25.551	318.668	31.047	388.790	20.854	312.715	30.892
14	340.357	26.000	310.520	31.330	340.872	20.667	391.994	30.903
15	355.882	25.984	326.119	30.962	333.075	20.838	338.533	30.669
16	354.112	25.606	298.752	31.268	382.122	21.077	323.176	30.957
17	356.443	25.439	333.644	30.832	330.655	21.521	335.786	30.604
18	375.528	26.369	327.466	30.931	344.084	20.684	358.240	30.643
19	343.011	25.743	365.515	30.954	381.801	20.959	403.214	30.692
20	363.223	29.961	326.742	31.109	383.786	20.518	327.936	30.813
21	381.830	25.477	334.483	31.360	324.960	20.912	339.297	30.657
22	393.052	25.684	380.316	30.982	399.307	20.816	336.141	30.835
23	360.850	25.954	328.024	31.365	367.281	20.895	303.323	30.536
24	376.653	25.543	402.972	30.989	367.724	20.755	335.486	30.543
25	400.658	24.860	347.278	31.112	358.709	20.843	323.060	30.851
Snitt	372.696	24.942	341.176	29.466	366.887	21.017	347.025	30.742
min	322.896	2.996	298.752	0.500	324.960	20.476	303.323	30.358
max	501.932	29.961	402.972	33.300	409.532	23.423	423.488	31.136

Tabell E.1: Tabell över simuleringsresultat av algoritm i MatLab. Avstånd är i längdenheter och tid i sekunder.

F

Pythonkod för objektsidentifieringssystem

```python
import cv2

stream = cv2.VideoCapture('URL')

while True:

    ret, frame = stream.read()
    cv2.imshow('Stream',frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

Listing F.1: Testning av videoströmmning till markstation utan objektsidentifiering

```python
import cv2
from darkflow.net.build import TFNet
import os


options = {
    'model': "cfg/tiny-yolo-voc-1c.cfg",
    'load': 59000,
    'threshold': 0.6
}

tfnet = TFNet(options)

img = cv2.imread("IMAGE_PATH", cv2.IMREAD_COLOR)

results = tfnet.return_predict(img)

for result in results:
    tl = (result['topleft']['x'], result['topleft']['y'])
    br = (result['bottomright']['x'], result['bottomright']['y'])
    label = result['label']
    confidence = format(result['confidence'],'.2f')
```

```
23      img = cv2.rectangle(img, tl, br, (0, 255, 0), 7)
24      img = cv2.putText(img, label + ": " + confidence, tl, cv2.
    FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
25
26  cv2.imshow("image", img)
27  cv2.namedWindow("image", 0)
28  cv2.resizeWindow("image", 1920, 1080)
29  cv2.imwrite("SAVE_PATH",img)
30
31  if(cv2.waitKey(0) & 0xFF == ord('q')):
32      cv2.destroyAllWindows()
```

Listing F.2: Objektsidentifieringstest på bildfil

```
1   import cv2
2   from darkflow.net.build import TFNet
3   import time
4   import os
5
6   options = {
7       'model': "cfg/tiny-yolo-voc-1c.cfg",
8       'load': 59000,
9       'threshold': 0.8 # Sets the desired threshold for detection.
10  }
11
12  tfnet = TFNet(options)
13
14  saveFrameDir = '/resultImages' # Set path to where processed images
        should be saved
15
16  cv2.namedWindow("img", 0)
17  width = 512
18  height = 288
19  cv2.resizeWindow("img", width, height)
20
21  stream = cv2.VideoCapture('URL')
22
23  n = 0
24  while (stream.isOpened()):
25      stime = time.time()
26      ret, img = stream.read()
27      if ret:
28
29          #Comment this if live feed of processed images is desired
30          cv2.imshow("img", img)
31
32          # Only process and save every 30th image due to slow computing.
33          # Change this according to the hardware used.
34          # If sufficent hardware is used and livestream is desired
35          # process every image (change to n%1) and also change the
        commented
36          # bits of code above and below.
37          if n%30 == 0:
38              results = tfnet.return_predict(img)
39              for result in results:
40                  tl = (result['topleft']['x'], result['topleft']['y'])
```

```
                br = (result['bottomright']['x'], result['bottomright'
    ]['y'])
                label = result['label']
                confidence = format(result['confidence']*100,'.0f')
                img = cv2.rectangle(img, tl, br, (0, 255, 0), 7)
                img = cv2.putText(img, label + ": " + confidence+"%",
    tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
                savePath = os.path.join(saveFrameDir, '{:06}.png'.
    format(n))

                #Uncomment this if live feed of processed images is
    desired
                #cv2.imshow("img", img)

                #Comment this if live feed of processed images is
    desired to save computing power.
                cv2.imwrite(savePath, img)
                print("Saving image number: " + n+"." + "        Save
    path: " + savePath)
     else:
         stream.release()
         break

     if cv2.waitKey(1) & 0XFF == ord('q'):
         cv2.destroyAllWindows()
         break

     n += 1
```

Listing F.3: Objektsidentifiering på videoström

```
import cv2
from darkflow.net.build import TFNet
import numpy as np
import os

option = {
    'model': 'cfg/tiny-yolo-voc-1c.cfg',
    'load': 59000,
    'threshold': 0.85,
    'gpu': 1.0
}

tfnet = TFNet(option)
videoDir = 'VIDEO_PATH'
saveFrameDir = 'resultVideos/'
capture = cv2.VideoCapture(videoDir)
width = int(capture.get(3))
height = int(capture.get(4))
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
fps = 29.97 #set to equal fps of video file
output = cv2.VideoWriter(saveFrameDir+'result_video_name.mp4', fourcc,
    fps, (width,height))

cv2.namedWindow("frame", 0)
cv2.resizeWindow("frame", width, height)
```

```
25
26 n = 0
27 while (capture.isOpened()):
28     ret, frame = capture.read()
29     if ret:
30         results = tfnet.return_predict(frame)
31         for result in results:
32             color = (0, 255, 0)
33             tl = (result['topleft']['x'], result['topleft']['y'])
34             br = (result['bottomright']['x'], result['bottomright']['y'
   ])
35             label = result['label']
36             confidence = format(result['confidence']*100,'.0f')
37             frame = cv2.rectangle(frame, tl, br, color, 7)
38             frame = cv2.putText(frame, label + ": " + confidence+"%",
   tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)
39         cv2.imshow('frame', frame) #comment for faster result but no
   rendering
40         print("Saving frame " + str(n))
41         n += 1
42         output.write(frame)
43         if cv2.waitKey(1) & 0xFF == ord('q'):
44             break
45     else:
46         break
47
48
49 # Release everything if job is finished
50 capture.release()
51 output.release()
52 cv2.destroyAllWindows()
```

Listing F.4: Objektsidentifiering på videofil

G

Matlab-kod för simuleringar

```
clc
clearvars
close all
tic
x_max = 200;
y_max = 200;
z_max = 200;

low1 = [120 100 45,120 150 45,170 150 45,170 100 45];
low2 = [70 140 45,70 190 45,120 190 45,120 140 45];
high = [70 140 100,70 190 100,120 190 100,120 140 100];
big = [50 50 45,50 150 45, 150 150 45, 150 50 45];

obs = {low1,low2};
%obs = {low1,high};
%obs = {big};
EPS = 20;
numNodes = 3000;

q_start.coord = [0 0 40];
q_start.cost = 0;
q_start.parent = 0;
q_goal.coord = [180 180 40];
q_goal.cost = 0;
z_plane = (q_goal.coord(3)-q_start.coord(3))/2;
z_plane2 = 40;


nodes(1) = q_start;
figure(1)
axis([0 x_max 0 y_max 0 z_max]),grid on
bar_plot([low1 low2],'green')
%bar_plot([low1 high],'green')
%bar_plot([big],'green')
hold on

plot3(q_start.coord(1),q_start.coord(2),q_start.coord(3),'b*','
    Linewidth', 3)
plot3(q_goal.coord(1),q_goal.coord(2),q_goal.coord(3),'r*','Linewidth',
     3)
hold on

```

```
for i = 1:1:numNodes
    q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max) z_plane2];
    %plot3(q_rand(1), q_rand(2), q_rand(3) , 'x', 'Color', [0 0.4470
    0.7410])

    % Pick the closest node from existing list to branch out from

    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist_3d(n.coord, q_rand);
        ndist = [ndist tmp];
    end
    [val, idx] = min(ndist);
    q_near = nodes(idx);

    q_new.coord = steer3d(q_rand, q_near.coord, val, EPS); %steer 3d
    noCollision = 1;
    for l = 1:1:length(obs)
        if noCollision3dnew(q_rand, q_near.coord, obs{l}) == 0
            noCollision = 0;
        end
    end
    if noCollision
        line([q_near.coord(1), q_new.coord(1)], [q_near.coord(2), q_new
    .coord(2)], [q_near.coord(3), q_new.coord(3)], 'Color', 'k', '
    LineWidth', 1);
        drawnow
        hold on
        q_new.cost = dist_3d(q_new.coord, q_near.coord) + q_near.cost;

        % Update parent to least cost-from node
        for j = 1:1:length(nodes)
            if nodes(j).coord == q_near.coord
                q_new.parent = j;
            end
        end

        % Append to nodes
        nodes = [nodes q_new];
        if q_new.coord == q_goal.coord
            break
        end
    end

end

D = [];
for j = 1:1:length(nodes)
    tmpdist = dist_3d(nodes(j).coord, q_goal.coord);
    D = [D tmpdist];
end

% Search backwards from goal to start to find the optimal least cost
    path
[val, idx] = min(D);
```

```
q_final = nodes(idx);
q_goal.parent = idx;
q_end = q_goal;
nodes = [nodes q_goal];
dist=0;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
    nodes(start).coord(2)], [q_end.coord(3), nodes(start).coord(3)], '
    Color', 'r', 'LineWidth', 2);
    hold on
    dist=dist+dist_3d(nodes(start).coord,q_end.coord);
    q_end = nodes(start);
end
dist
toc
```

Listing G.1: Objektsidentifiering på videofil

```
clc
clearvars
close all
tic
x_max = 200;
y_max = 200;
z_max = 200;

low1 = [120 100 45,120 150 45,170 150 45,170 100 45];
low2 = [70 140 45,70 190 45,120 190 45,120 140 45];
low3 = [50 50 45,50 100 45,100 100 45,100 50 45];
low4 = [100 100 45,100 150 45,150 150 45,150 100 45];
high = [70 140 100,70 190 100,120 190 100,120 140 100];
big = [50 50 45,50 150 45, 150 150 45, 150 50 45];
coll = [120 100 50,120 150 50,170 150 50,170 100 50];

obs = {low1,low2};
%obs = {low1,high};
%obs = {big};
%obs = {coll};
%obs = {low3,low4};

EPS = 20;
numNodes = 3000;

q_start.coord = [0 0 40];
q_start.cost = 0;
q_start.parent = 0;
q_goal.coord = [180 180 40];
q_goal.cost = 0;
z_plane = (q_goal.coord(3)-q_start.coord(3))/2;
z_plane2 = 40;


nodes(1) = q_start;
figure(1)
axis([0 x_max 0 y_max 0 z_max]),grid on
```

```
bar_plot([low1 low2],'green')
%bar_plot([low3 low4],'green')
%bar_plot([low1 high],'green')
%bar_plot([big],'green')
%bar_plot([coll],'green')
hold on

plot3(q_start.coord(1),q_start.coord(2),q_start.coord(3),'b*','Linewidth', 3)
plot3(q_goal.coord(1),q_goal.coord(2),q_goal.coord(3),'r*','Linewidth', 3)
hold on

for i = 1:1:numNodes
    q_rand = [floor(rand(1)*x_max) floor(rand(1)*y_max) z_plane2];

    % Pick the closest node from existing list to branch out from
    ndist = [];
    for j = 1:1:length(nodes)
        n = nodes(j);
        tmp = dist_3d(n.coord, q_rand);
        ndist = [ndist tmp];
    end
    [val, idx] = min(ndist);
    q_near = nodes(idx);

    q_new.coord = steer3d(q_rand, q_near.coord, val, EPS); %steer 3d
    noCollision = 1;
    noCollisionUp = 1;
    for l = 1:1:length(obs)
        if noCollision3dnew(q_rand, q_near.coord, obs{l}) == 0
            noCollision = 0;
        end
    end

    % if collision, check if the space above is clear, and branch up
    if noCollision == 0
        noCollisionUp = 1;
        q_temp=q_rand;
        q_temp(3)=z_plane2+15;
        for m = 1:1:length(obs)
            if noCollision3dnew(q_temp, q_near.coord, obs{m}) == 0
                noCollisionUp = 0;
            end
        end
        if noCollisionUp
            q_new.coord = steer3d(q_temp, q_near.coord, val, EPS);
        end
    end

    if noCollisionUp
        line([q_near.coord(1), q_new.coord(1)], [q_near.coord(2), q_new.coord(2)], [q_near.coord(3), q_new.coord(3)], 'Color', 'k', 'LineWidth', 1);
        drawnow
        hold on
```

```
        q_new.cost = dist_3d(q_new.coord, q_near.coord) + q_near.cost;

        % Update parent to least cost-from node
        for j = 1:1:length(nodes)
            if nodes(j).coord == q_near.coord
                q_new.parent = j;
            end
        end

        % Append to nodes
        nodes = [nodes q_new];
        if q_new.coord == q_goal.coord
            break
        end
    end

end

D = [];
for j = 1:1:length(nodes)
    tmpdist = dist_3d(nodes(j).coord, q_goal.coord);
    D = [D tmpdist];
end

% Search backwards from goal to start to find the optimal least cost
    path
[val, idx] = min(D);
q_final = nodes(idx);
q_goal.parent = idx;
q_end = q_goal;
nodes = [nodes q_goal];
dist=0;
while q_end.parent ~= 0
    start = q_end.parent;
    line([q_end.coord(1), nodes(start).coord(1)], [q_end.coord(2),
    nodes(start).coord(2)], [q_end.coord(3), nodes(start).coord(3)], '
    Color', 'r', 'LineWidth', 2);
    hold on
    dist=dist+dist_3d(nodes(start).coord,q_end.coord);
    q_end = nodes(start);
end
dist
toc
```

Listing G.2: Objektsidentifiering på videofil

```
function bar_plot(a,color)
% BAR_PLOT plots a 3D Bar from 0 och the Z-axis to the points included
    in
% "a".
% The first 3 value in a are categorized as ONE point (X,Y,Z)
% The first 4 points in a are used to create a BAR from those 4 points
    and
% down to the xy-plane.
% The order of point in a matters and are as follows! Another order
    might
```

```
% work but is not recommended.
% "a" =[lowLeft, highLeft, highRight, lowRight];
% Several figures can be plottet from the same vector. "a" can have 4*N
% ints, where N is number of bars.
% INPUTS:
% a = vector with koordinates for bar
% OUPUTS:
% Plot figure(s) in the form of bar(s).
%
% EXAMPLES
% bar_plot(a,color)
% where "a" is a vector of lenght 3*4*N
%

% ------------------------------------Code Starts Here
    ------------------------------- %
% Define the vertexes of the unit cubic

s = size(a);
numbOfObstacles = s(2)./3./4;
for i = 0:numbOfObstacles-1
    shiftObs = i * 12;
p1 =a((1 + shiftObs):(3 + shiftObs));
p2 =a((4 + shiftObs):(6 + shiftObs));
p3 =a((7 + shiftObs):(9 + shiftObs));
p4 =a((10 + shiftObs):(12 + shiftObs));
p11 =[a((1 + shiftObs):(2 + shiftObs)) 0];
p12 =[a((4 + shiftObs):(5 + shiftObs)) 0];
p13 =[a((7 + shiftObs):(8 + shiftObs)) 0];
p14 =[a((10 + shiftObs):(11 + shiftObs)) 0];

v =[p1;p2;p3;p4];
v2 =[p11;p12;p13;p14];
v3 =[p1;p2;p12;p11];
v4 =[p3;p4;p14;p13];
v5 =[p2;p3;p13;p12];
v6 =[p4;p1;p11;p14];
f = [1 2 3 4];

patch('Faces',f,'Vertices',v,'FaceColor',color)
patch('Faces',f,'Vertices',v2,'FaceColor',color)
patch('Faces',f,'Vertices',v3,'FaceColor',color)
patch('Faces',f,'Vertices',v4,'FaceColor',color)
patch('Faces',f,'Vertices',v5,'FaceColor',color)
patch('Faces',f,'Vertices',v6,'FaceColor',color)
alpha(.5)
end


end
% ------------------------------------Code Ends Here
    ------------------------------- %
```

Listing G.3: Objektsidentifiering på videofil

```
function val = ccw(A,L,M)
```

```
    val = (M(2)-A(2)) * (L(1)-A(1)) > (L(2)-A(2)) * (M(1)-A(1));
end
```

Listing G.4: Objektsidentifiering på videofil

```
function d = dist_3d(q1,q2)
    d = sqrt((q1(1)-q2(1))^2 + (q1(2)-q2(2))^2 + (q1(3)-q2(3))^2);
end
```

Listing G.5: Objektsidentifiering på videofil

```
function nc = noCollision3dnew(n2, n1, o)
    A = [n1(1) n1(2) n1(3)];
    B = [n2(1) n2(2) n2(3)];
    obs = [o(1) o(2) o(3) o(7) o(5) o(3)];
    %if one of the nodes are within the obstacle, returns 0
    if o(3) < n2(3)
        nc = 1;
    elseif (n1(3) <= obs(3) && n1(1) >= obs(1) && n1(1) <= obs(4)  &&
    n1(2) >= obs(2) && n1(2) <= obs(5))...
        ||  (n2(3) <= obs(3) && n2(1) >= obs(1) && n2(1) <= obs(4)  &&
    n2(2) >= obs(2) && n2(2) <= obs(5))
        nc = 0;
    else
        C1 = [obs(1),obs(2)];
        D1 = [obs(1),obs(5)];
        C2 = [obs(1),obs(2)];
        D2 = [obs(4),obs(2)];
        C3 = [obs(4),obs(5)];
        D3 = [obs(4),obs(2)];
        C4 = [obs(4),obs(5)];
        D4 = [obs(1),obs(5)];

        % Check if path from n1 to n2 intersects any of the four edges
    of the
        % obstacle

        ints1 = ccw(A,C1,D1) ~= ccw(B,C1,D1) && ccw(A,B,C1) ~= ccw(A,B,
    D1);
        ints2 = ccw(A,C2,D2) ~= ccw(B,C2,D2) && ccw(A,B,C2) ~= ccw(A,B,
    D2);
        ints3 = ccw(A,C3,D3) ~= ccw(B,C3,D3) && ccw(A,B,C3) ~= ccw(A,B,
    D3);
        ints4 = ccw(A,C4,D4) ~= ccw(B,C4,D4) && ccw(A,B,C4) ~= ccw(A,B,
    D4);
        if ints1==0 && ints2==0 && ints3==0 && ints4==0
            nc = 1;
        else
            nc = 0;
        end
    end
end
```

Listing G.6: Objektsidentifiering på videofil

```
function A = steer3d(qr, qn, val, eps)
```

```
    qnew = [0  0];
    if  val >= eps
        qnew(1) = qn(1) + ((qr(1)-qn(1))*eps)/dist_3d(qr,qn);
        qnew(2) = qn(2) + ((qr(2)-qn(2))*eps)/dist_3d(qr,qn);
        qnew(3) = qn(3) + ((qr(3)-qn(3))*eps)/dist_3d(qr,qn);
    else
        qnew(1) = qr(1);
        qnew(2) = qr(2);
        qnew(3) = qr(3);
    end

    A = [qnew(1), qnew(2), qnew(3)];
end
```

Listing G.7: Objektsidentifiering på videofil