CHALMERS | GÖTEBORGS UNIVERSITET

Självnavigerande bevingad drönare

Utveckling och implementering av autonom navigation samt kommunikation för flygplansliknande drönare

Kandidatarbete vid institutionen för Data- och informationsteknik

NIKLAS BAERVELDT, ALGOT JOHANSSON, ANTHONY KALCIC,
JOEL LÖFVING, RIKARD TEODORSSON & HENRIK WIBERG

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

Kandidatarbete DATX02-19-13

Självnavigerande bevingad drönare

Utveckling och implementering av autonom navigation samt kommunikation för flygplansliknande drönare

NIKLAS BAERVELDT
ALGOT JOHANSSON
ANTHONY KALCIC
JOEL LÖFVING
RIKARD TEODORSSON
HENRIK WIBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och informationsteknik
Chalmers tekniska högskola

Självnavigerande bevingad drönare
Utveckling och implementering av autonom navigation samt kommunikation för flygplansliknande drönare

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

Kandidatarbete DATX02-19-13
Institutionen för Data- och informationsteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Telephone +46 31 772 1000

Typeset in LaTeX
Göteborg, Sverige 2019

Self-navigating fixed-winged drone
Development and implementation of autonomous navigation and communication for fixed-wing drones

NIKLAS BAERVELDT
ALGOT JOHANSSON
ANTHONY KALCIC
JOEL LÖFVING
RIKARD TEODORSSON
HENRIK WIBERG
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The consumer market has during the recent years seen a significant rise in the number of smaller unmanned aircrafts, so-called drones. With their capability to be used for many different tasks, such as 3D-photography, surveillance and search-and-rescue operations, drones have proven to be a valuable resource. Making them autonomous further expands their area of use.

As a contribution for future advancement, this paper describes the development and implementation process for autonomous navigation and long-range communication for fixed-wing drones.

Using a Raspberry PI microcontroller, PID controllers for stabilisation and autonomous navigation, techniques such as sensor fusion using Kalman filtering to improve GPS accuracy, altitude measurements using a barometer, and system modelling for simulations, the developed software is integrated on hardware and tested on a small-scale model aircraft.

The presented result is a functional system for autonomous navigation and communication implemented on a drone; capable of navigating between a given set of waypoints, maintaining altitude and speed. Communication is established using an Android app and an external database as a mediator between the drone and clients to enable real-time telemetry updates and as a reliable connection over the internet.

These results are satisfactory as the drone can both take-off and navigate as well as return to its launch position fully autonomously, and at the same time communicate its status to operators. However, the project is not successful in implementing an autonomous landing sequence, which requires manual operation. This, along with collision avoidance are suggestions for future work.

Swedish is used throughout the paper.

Keywords: drone, airplane, autonomous, self-steering, navigation, real-time telemetry

Självnavigerande bevingad drönare
Utveckling och implementering av autonom navigation samt kommunikation för flygplansliknande drönare

NIKLAS BAERVELDT
ALGOT JOHANSSON
ANTHONY KALCIC
JOEL LÖFVING
RIKARD TEODORSSON
HENRIK WIBERG

Institutionen för Data- och informationsteknik
Chalmers Tekniska Högskola

Sammandrag

Konsumentmarknaden har under de senaste åren sett en signifikant ökning i antalet obemannade flygfarkoster, så kallade drönare. Med en förmåga att användas för många olika uppgifter, så som 3D-fotografering, övervakning och sök- och räddningsuppdrag, har drönare visat sig vara en värdefull resurs. Att göra dem autonoma ökar deras användningsområden ytterligare.

Som ett bidrag till framtida utveckling beskriver den här rapporten utvecklings- och implementeringsprocessen för autonom navigation och långdistanskommunikation för bevingade drönare.

Med hjälp av en Raspberry PI mikrokontroller, PID-regulatorer för stabilisering och autonom navigation, tekniker så som sensorfusion med Kalmanfiltrering för att förbättra GPS-noggrannheten, höjdmätningar med hjälp av en barometer och systemmodellering för simuleringar, är den utvecklade mjukvaran integrerad på hårdvara och testad på ett mindre modellflygplan.

Det presenterade resultatet är ett fungerande system för autonom navigation och kommunikation implementerad på en drönare; kapabel att navigera mellan en given lista med vägpunkter, bibehållande sin höjd och hastighet. Kommunikation är etablerad med en Android-applikation och en extern databas som medlar mellan drönaren och klienter för att möjliggöra realtidstelemetri och som en pålitlig anslutning över internet.

Dessa resultat är tillfredsställande då drönaren både kan lyfta och navigera samt återvända till sin startposition autonomt, och samtidigt kommunicera sin status till operatör. Projektet lyckades dock inte med att genomföra en autonom landningssekvens, som kräver manuell styrning. Detta, tillsammans med kollisionsundvikning, är förslag till framtida arbete.

Nyckelord: drönare, flygplan, autonom, självstyrande, navigation, realtidstelemetri

Ordlista

Attityd - Drönarens vinkel i det tredimensionella rummet.

Börvärde - Önskad utsignal från ett system inom reglertekniken. Referenssignal.

Flygväg - Flera vägpunkter i en specifik ordning utgör tillsammans en flygväg.

Inkapsling - Att metoderna i en modul (se nedan) utgör ett gränssnitt (API) för andra moduler i mjukvaran utan att avslöja hur de fungerar.

Kompilator - En kompilator kontrollerar att programkod inte bryter mot ett programspråks grammatiska regler och därefter översätter programspråket till körbar kod.

Kompilering - Körning av kompilatorn.

Modul - En ensam kodklass eller en sammansättning av flera klasser som tillsammans utför en funktion i mjukvaran.

Pitch - Vridande rörelse kring flygplanets tväraxel (Y-axeln i figur 2.1).

Roll - Vridande rörelse kring flygplanets längdaxel (X-axeln i figur 2.1).

Servo - En liten motor som styr olika komponenter på flygplanet, exempelvis roder.

Vägpunkt - En geografisk koordinat att navigera mot.

Yaw - Vridande rörelse kring flygplanets lodaxel (Z-axeln i figur 2.1).

Ärvärde - Uppmätt utsignal från ett system inom reglertekniken.

Innehåll

Figurer

Tabeller

1

Introduktion

Under de senaste åren har marknaden sett en ökning i användandet och tillgängligheten av mindre flygfarkoster, så kallade drönare, både genom ökat utbud och lägre pris [1]. Drönare används både för hobbyflygning bland privatpersoner och i kommersiella syften. De kan till exempel användas för fotografering, 3D-scanning av stora byggnader och livräddningsuppdrag i svårtillgängliga områden [2]. Även ett ökande intresse för drönare inom logistik kan urskiljas [3][4][5] som ett potentiellt effektivare, flexiblare och ett mer miljövänligt sätt att transportera varor. Inspektion av storskalig infrastruktur, till exempel elnät, har visat sig kunna utföras av med drönare [6].

Kommersiella drönare har vanligtvis en flygtid på maximalt 30 minuter med en flygdistans mellan en till två mil [7]. I dagsläget fokuserar många utvecklare på att öka räckvidd, hastighet och flygtid, men också på att öka autonomiteten hos drönare, vilket kan gagna kommersiella syften. Möjligheterna kan även expanderas till att täcka privatpersoners användning av drönare i hobbysyften eller som en del av vardagen. I längden skulle autonoma drönare kunna bistå med smidigare logistiklösningar än de vi har idag, bättre övervakning av, till exempel, stora värdebindande strukturer och hjälp vid katastrofhantering eller sök- och räddningsuppdrag.

Dock riskerar denna teknologi att inskränka människors privatliv genom, bland annat, övervakning. Detta diskuteras vidare i avsnitt 1.6.

1.1 Definition av drönare

”Drönare” är ett namn som används för en ”förarlös flygfarkost för militärt eller civilt bruk” [8]. Det finns utöver detta två huvudsakliga indelningar av drönare vad gäller flygkonfiguration. I boken *Domesticating Drones: The Technology, Law and Economics of Unmanned Aircraft* [9] nämns *fixed-wing* och *rotor-wing drones*, där förstnämnda syftar till luftfarkoster som genererar lyft med statiska vingar medan den senare syftar till luftfarkoster som genererar lyft med rotorblad. Fixed-wing drones är den typ av drönare som det här projektet fokuserar på, och kommer utöver ”drönare” även benämnas ”flygplan” i rapporten.

1.2 Gruppens studiebakgrund

Projektet har genomförts av sex studenter med olika kompetenser från civilingenjörsprogrammen *Automation & Mekatronik*, *Datateknik* samt *Informationsteknik* vid Chalmers Tekniska Högskola. Det föll sig därför naturligt att projektet inriktades åt mjukvaruhållet, där resultaten implementerats och testats på hårdvara.

1.3 Syfte

Projektet har till syfte att skapa och implementera mjukvara för autonom styrning och långdistanskommunikation för lätta bevingade luftfarkoster. Mjukvaran ska kunna ligga till grund för framtida utveckling och forskning som ämnar nyttja autonoma drönare.

1.4 Mål

Det övergripande problemet innefattar att, enligt syftet, utveckla autonom styrning för drönare och ge möjlighet till kommunikation av kommandon och telemetri med detta system. Detta har sedan brutits ner i mindre, mer överskådliga mål, här listade i prioriteringsordning:

- Kommunikation till och från drönaren
- Navigation mellan olika vägpunkter
- Generell och anpassningsbar mjukvara
- Lyfta från marken
- Landa
- Undvika geografiskt definierat område

Enligt lagkrav måste det finnas en operatör som har drönaren inom synhåll och som kan ta kontrollen över och styra den manuellt i alla givna situationer om inte operatören har speciella tillstånd för detta [10]. Alltså behöver styrning med handkontroll implementeras. Därtill bör även ett system för kommunikation av vägpunkter, till vilka den autonoma styrningen ska navigera, upprättas på så vis att operatör trådlöst kan ge kommandon innan flygning påbörjas. Systemet ska också kommunicera drönarens status kontinuerligt under flygningen. Ett delproblem till detta är att kommunikationen bör kunna hanteras med ett visuellt gränssnitt för användaren. Under flygning ska drönaren, utan mänskligt ingripande, kunna navigera mellan vägpunkter förutbestämd av operatör

Mjukvaran som utvecklas ska vara så pass generell att den kan användas för olika bevingade drönare. Detta för att tillåta fortsatt utveckling och forskning på ämnet med det här projektet som bas.

Flygplanet ska kunna starta själv efter kommando från operatör. Fysiskt skulle detta kunna ske med extra tillförd kraft, till exempel från katapult. Dock ska flygplanet reglera sig själv under stigning.

Efter slutförd flygning ska drönaren landa inom ett fördefinierat område. Landningen, som alla föregående delar av flygningen, ska inte kräva manuell kontroll. Själva marksättningen ska ske kontrollerat så att drönaren inte tar skada och kan användas strax efteråt för en ny flygning, givet att batteriet inte är urladdat.

Drönaren ska kunna undvika fördefinierade zoner så som restriktionsområden och kontrollzoner, även om flygvägen passerar igenom området. Den ska då bryta från den planerade flygvägen för att undvika zonen. När drönaren har passerat zonen ska den återuppta den planerade flygvägen. Dessa avgränsade zoner ska kunna definieras av användaren både före och under flygning.

1.5 Avgränsningar

I följande avsnitt presenteras kort de avgränsningar som satts för projektet.

Som prototyp kommer ett befintligt modellflygplan att användas. Projektet avgränsas alltså från att fysiskt utforma flygplanet, och fokuserar istället på utveckling av mjukvara implementerad på hårdvara som monteras på modellflygplanet.

Även om projektet har som mål att designa all mjukvara så generell som möjligt så är det mycket svårt att göra den hårdvarunära koden generell. I kapitel 5.1 beskrivs hur arkitekturen delas upp i fyra olika ”lager”, där det lägsta är närmast hårdvaran. Projektet avgränsas till att endast göra de två översta lagren helt generella, och lämnar de två mest hårdvarunära lagren att modifieras för varje enskild drönare och dess hårdvara.

1.6 Samhälleliga och etiska aspekter

Mjukvaran som skapas i det här projektet är tänkt att ligga till grund för teknisk utveckling och forskning som ämnar nyttja autonoma drönare. Projektet har således potential för en utbredd påverkan på samhället.

Ett vanligt användningsområde hos drönare är övervakning av olika slag [6], [11], vilket kan ha oönskad effekt. Ökad tillgänglighet till drönare eller bättre drönarteknologi skulle därför kunna medföra ökad övervakning och därmed minskad integritet. Den ökade tillgängligheten till drönare skulle även kunna orsaka oönskat buller i tätbebyggda områden.

Det finns dock även positiva samhällsaspekter. Autonoma drönare har potential att bistå vid sök- och räddningsuppdrag, med hjälp vid naturkatastrofer, samt inspektion av större infrastrukturer. Autonomiteten hos en drönare kan även medföra att företag kan spara in på personalkostnader och samtidigt få snabbare leveranser inom logistik.

I slutändan anser gruppen att de positiva samhällsaspekterna med fortsatt drönarutveckling väger tyngre än de negativa.

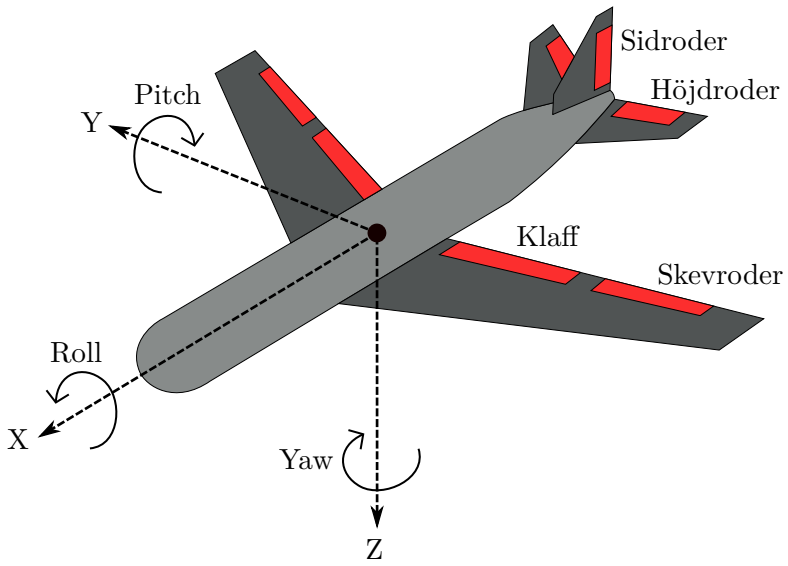
2

Teori

Detta avsnitt behandlar teori gällande olika moment som är relevanta för projektet och ligger som underlag till resten av rapporten.

2.1 Attityd

En luftfarkosts orientering i luften kallas *attityd* och beskrivs med tre vinklar: *yaw*, *pitch* och *roll*. Yaw beskriver farkostens riktning horisontellt (vinkel från norr), pitch beskriver hur mycket nosen pekar upp eller ner och roll dess lutning i sidled [12], se figur 2.1. Svenskspråkig terminologi för att beskriva orientering finns (gir, tipp respektive roll) [13], men den används sällan till förmån för den engelska. Därför kommer den engelska terminologin användas i denna rapport.



Figur 2.1: Ett flygplan med *yaw*, *pitch*, *roll* och sina roder utritade. Notera att rodren är symmetriska och därmed är inte namnen för vänster sida av planet utsatta.

2.1.1 Rotationsmatriser för yaw, pitch och roll

En rotation med yaw av α, pitch av β eller roll av γ beskrivs av följande rotationsmatriser:

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.1) \qquad R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \qquad R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$

Om dessa multipliceras erhålls en ensam rotationsmatris vilken beskriver farkostens *totala* riktning [14]:

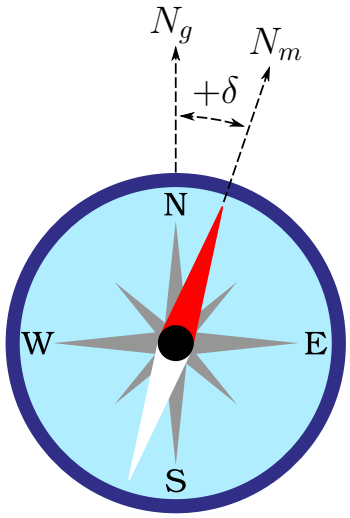
$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma) = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix} \quad (2.2)$$

2.2 Magnetisk deklination

En kompass använder sig av det magnetiska fältet för att bestämma nordriktningen. Dessvärre pekar inte magnetisk nord exakt mot den geografiska nordpolen, eftersom den magnetiska nordpolen förflyttar sig med tiden. Vinkeln mellan riktningarna mot dessa punkter kallas den magnetiska deklinationen [15], se figur 2.2.

Ett problem med detta är att sensorn som beräknar flygplanets yaw gör det i relation till den magnetiska nordpolen och inte den geografiska. Detta gör att man måste räkna bort deklinationen för att få yaw i relation till den verkliga nordpolen.

För att räkna bort magnetisk deklination ersätts yaw i matris 2.1 med deklinationen och multipliceras sedan med en vektor bestående av planets yaw, pitch och roll (i den ordningen). Detta resulterar i en ny vektor av attityden, fast med den magnetiska deklinationen borträknad.

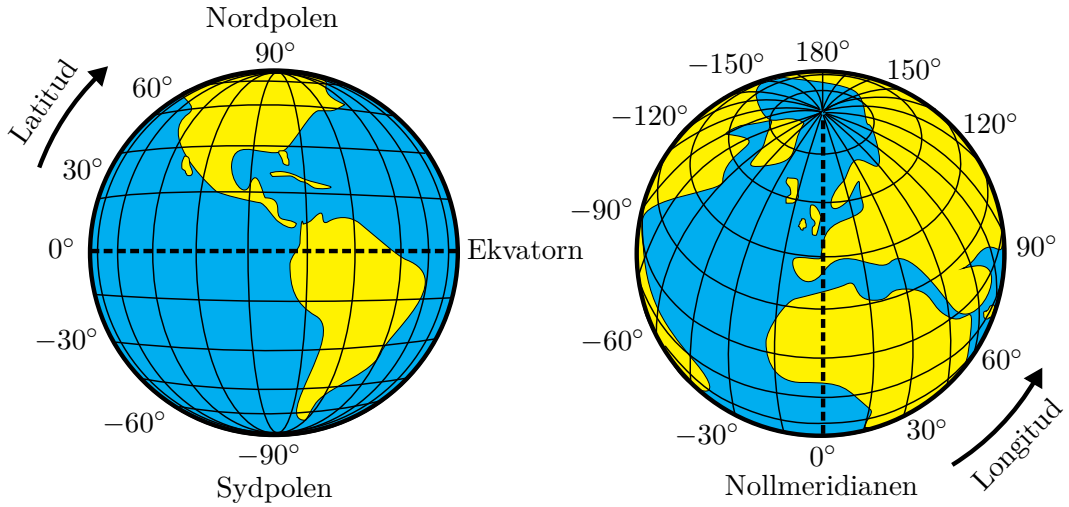


Figur 2.2: Visualisering av magnetisk deklination av δ mellan den verkliga (N_g) och den magnetiska (N_m) nordriktningen.

2.3 Positionsbestämning

Ett system för att bestämma position är GPS, Global Positioning System [16]. Position från systemet kan ges i form av vinklar kallade *latitud* och *longitud*. Latitud beskriver positionen i nord- och sydriktning relativt till ekvatorn medan longitud beskriver positionen i öst- och västriktningarna relativt till nollmeridianen, vilket kan ses i figur 2.3.

Under flygning är det viktigt att inte ha för låg noggrannhet på positionen då det kan försämra precisionen hos navigationsuppdrag. Typiska GPS-mottagare har en precision på cirka 5-20 meter [17]. Precisionen minskar dock inomhus [18].



Figur 2.3: Hur latitud och longitud fungerar. Från [19], lätt redigerad, CC0.

2.4 Sensorfusion

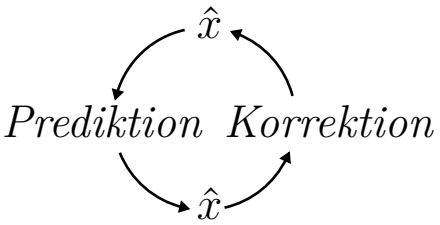
Sensorers mätvärde innehåller alltid en del så kallat ”brus”, en osäkerhet som gör att värdet man uppmätt inte ger en exakt bild av verkligheten [20]. Detta kan antingen bero på externa störningar eller på att noggrannheten hos sensorn är för låg [20].

För att samla mer precisa värden ur en mätning, reducera brus och öka precisionen kan så kallad *sensorfusion* användas [21]. Sensorfusion kombinerar värden från olika mätningar eller estimeringar för att uppskatta ett mer exakt mätvärde som bör vara närmare verkligheten.

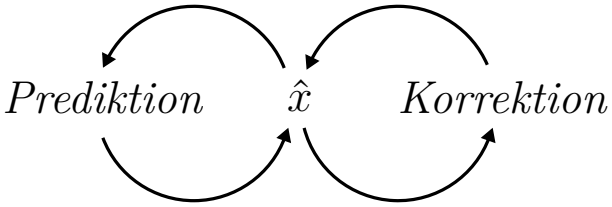
2.4.1 Kalmanfilter

Ett Kalmanfilter är en typ av sensorfusion [22] som sammanfogar två eller flera mätningar av ett systems tillstånd till en estimering som är mer tillförlitlig än varje enskild mätning [23]. Kalmanfiltret består av två steg: *prediktion* och *korrektion* [22]. I prediktionssteget estimeras systemets tillståndsvariabler ($\hat{x}$) och dess osäkerheter baserat på föregående tillstånd samt en matematisk modell av systemet [22].

I korrektionssteget gör systemet sedan en absolut observation av tillståndet, vilken inte beror på det tidigare tillståndet. Dock kombineras den sedan med det tidigare tillståndet från prediktionssteget för att få en mer precis estimering av sitt tillstånd [22]. I detta fallet



Figur 2.4: Visualisering av hur ett Kalmanfilter för tillståndsvariblerna i $\hat{x}$ fungerar med ett prediktionssteg för varje observation. Detta används för att få bättre estimeringar av $\hat{x}$ vid ny observation genom att även ta hänsyn till prediktionen.



Figur 2.5: Visualisering av hur ett Kalmanfilter för tillståndsvariblerna i $\hat{x}$ fungerar med prediktionsstegen oberoende av korrektionsstegen. Detta innebär att man kan göra flera estimeringar av tillståndet innan man får en observation av det. Även här blir de slutliga estimeringarna bättre efter korrektionsstegen.

sker prediktionssteget direkt efter observationssteget. Se figur 2.4 för en visualisering av prediktions- och korrektionssteget med en prediktion per korrektion.

Ett Kalmanfilter kan även genomgå prediktionsstegen oberoende av korrektionsstegen. Detta kan användas för att få flera estimeringar mellan varje observation. Det kan lösa problem som till exempel att observationer av tillståndet inte sker med tillräckligt hög frekvens [24].

Ett annat problem som denna variant löser är då källan för observationerna är osäker och det finns en risk att de försvinner under vissa perioder. Där kan det vara bra att ändå kunna estimera tillståndet [22]. Se figur 2.5 för en visualisering av hur denna variant av Kalman går till.

Estimeringarna är inte absoluta utan uppdaterar systemets tillstånd i relation till det föregående tillståndet. Därmed ökar osäkerheten för varje estimering systemet gör [22].

I prediktionssteget sker följande beräkningar:

$$\hat{\mathbf{x}}_k = \mathbf{F}_k\hat{\mathbf{x}}_{k-1} + \mathbf{B}_k\overrightarrow{\mathbf{u}_k}$$
$$\mathbf{P}_k = \mathbf{F_k}\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k$$

där

$\hat{\mathbf{x}}$ är en vektor av systemets tillståndvariabler.

$\mathbf{F}$ är *state-transition*-modellen för systemet

$\mathbf{B}$ är kontrollmatrisen.

$\overrightarrow{\mathbf{u}}$ är kontrollvektorn.

$\mathbf{P}$ är kovariansmatrisen för $\hat{\mathbf{x}}$

$\mathbf{Q}$ är kovariansen för processbrus.

Prediktionen följer en normaldistribution:

$$\mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\mathbf{H}_k\hat{\mathbf{x}}_k, \mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T)$$

där

$\boldsymbol{\mu}_0$ är medelvärdesvektorn av prediktionen.

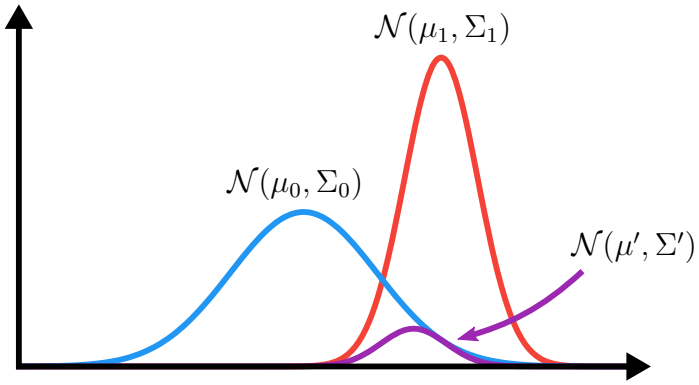
$\boldsymbol{\Sigma}_0$ är kovariansmatrisen för prediktionen.

$\mathbf{H}$ är en matris för att konvertera estimeringen till samma enheter som observationen.

I korrektionssteget används prediktionsvärdena från ovan tillsammans med de observerade tillståndsvariablerna $\overrightarrow{\mathbf{z}}$, samt deras kovariansmatris $\mathbf{R}$. Även dessa följer en normaldistribution:

$$(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) = (\overrightarrow{\mathbf{z}_k}, \mathbf{R}_k)$$

Multiplicerar man de observerade tillståndsvariablerna med de estimerade tillståndsvariablerna får man en ny normaldistribution $\mathcal{N}(\boldsymbol{\mu}', \boldsymbol{\Sigma}')$. Hur detta kan se ut i en av dimensionerna visualiseras i figur 2.6.



Figur 2.6: Visualisering av två endimensionella normaldistributioner $\mathcal{N}(\mu_0, \Sigma_0)$ och $\mathcal{N}(\mu_1, \Sigma_1)$ samt deras produkt $\mathcal{N}(\mu', \Sigma')$.

De slutgiltiga ekvationerna för att sätta $\hat{\mathbf{x}}_k$ och $\mathbf{P}_k$ till deras nya värden, $\boldsymbol{\mu}'$ och $\boldsymbol{\Sigma}'$, ser ut som följer:

$$\begin{aligned} \mathbf{K}' &= \mathbf{P}_k\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R}_k)^{-1} \\ \hat{\mathbf{x}}_k' &= \hat{\mathbf{x}}_k + \mathbf{K}'(\overrightarrow{\mathbf{z}_k} - \mathbf{H}_k\hat{\mathbf{x}}_k) \\ \mathbf{P}_k' &= \mathbf{P}_k - \mathbf{K}'\mathbf{H}_k\mathbf{P}_k \end{aligned} \tag{2.3}$$

2.4.2 Kalmanfilter för accelerometer och GPS

I projektet kommer Kalmanfiltrering användas för positionsestimering genom att kombinera GPS och accelerometerdata. Grundtanken är att varje gång en GPS-position avläses konverteras den först till meter och med hjälp av den föregående GPS-positionen beräknas en hastighet. (Genom att ta sträckan genom tiden.) Därefter jämförs dessa med den position och hastighet som är beräknad av accelerometern. Slutligen sker en estimering av det mest troliga tillståndet.

Fram till att en ny GPS-position avläses approximerar systemet nästkommande positioner och hastigheter med hjälp av accelerometern. Eftersom denna har högre uppdateringsfrekvens än GPS-sensorn kommer fler positionsutläsningar kunna ske.

2.4.3 Beräkning av Q och R för Kalmanfiltret

Det är generellt svårt att beräkna bra värden för $\mathbf{Q}$ och $\mathbf{R}$ [25][26].

En metod för att bestämma $\mathbf{Q}$ och $\mathbf{R}$ för fallet med positionsestimering är att sätta variansen för accelerometern diagonalt i $\mathbf{Q}$ och variansen för GPS-sensorn diagonalt i $\mathbf{R}$

[24]. En annan metod är att uppskatta deras värden [27].

2.4.4 Acceleration i kardinalriktningar

För att kunna sammanfoga accelerationen med GPS-värdena krävs det att man konverterar accelerationen ($\mathbf{a}$) från accelerometern i planet till acceleration i kardinalriktningarna ($\mathbf{a}_c$). Detta görs genom att multiplicera $\mathbf{a}$ med rotationsmatrisen i ekvation 2.2 och sedan räkna bort den magnetiska deklinationen enligt avsnitt 2.2.

2.4.5 Latitud och longitud i meter

För att kunna sammanfoga GPS-värdena (latitud- och longitudvinklar) med accelerometervärdena ($\mathrm{m/s^2}$) måste någon av dem konverteras så att de får kompatibla enheter. Ett sätt att åstadkomma detta är att konvertera GPS-värdena till meter [24].

Om man utgår ifrån att jorden är en perfekt sfär kan detta göras genom att inledningsvis implementera en funktion för att få en vinkel mellan två koordinater på sfären, se ekvation 2.4 samt figur 2.7. Notera att koordinaternas position definieras med vinklar på samma sätt som GPS-koordinater (se figur 2.3).

$$\Delta\sigma = f((\phi_1, \lambda_1), (\phi_2, \lambda_2)) = \arccos\left(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda)\right) \tag{2.4}$$

Därefter kan dessa vinklar konverteras till meter genom att multiplicera dem med c_1 från ekvation 2.5 där täljaren är jordens medelomkrets [28]. För att konvertera tillbaka dem kan man sedan multiplicera dem med c_2 från ekvation 2.6.

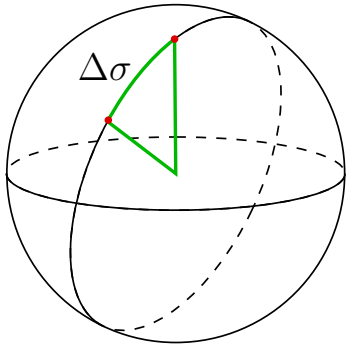
För att slutligen få latituden och longituden i meter var för sig kan man använda $c_1 \cdot f((\phi, 0), (0, 0))$ och $c_1 \cdot f((0, \lambda), (0, 0)) \cos\phi$ respektive.

$$c_1 = \frac{40030822.7}{360} \tag{2.5}$$

$$c_2 = \frac{360}{40030822.7} \tag{2.6}$$

Jorden är inte är helt sfärisk utan buktar ut lite vid ekvatorn [29]. Detta innebär att om man använder f för att konvertera longituden till meter, sedan lägger till en meter, och därefter konverterar tillbaka till longituden så har man egentligen gått lite längre än en meter. För latituden blir det lite mindre. Dock är jordens radie vid ekvatorn 6378.137 km medan radien till jordens poler är 6356.752 km [30]. Detta innebär att skillnaden på radien och därmed omkretsen är cirka 0.34% (se ekvation 2.7). Detta anses tillräckligt lite för att inte ha någon inverkan på positionsestimeringen och eftersom tillståndet konstant jämförs med ett absolut GPS-värde kommer inte felet ackumulera. Därmed anses approximationen fungera för ändamålet.

$$\frac{6378.137}{6356.752} \approx 1.0034 \tag{2.7}$$



Figur 2.7: En vinkel ($\Delta\sigma$) mellan två koordinater på en sfär.

2.5 Reglerteknik

Styrsystem brukar kategoriseras på två olika sätt, öppen respektive återkopplad styrning [31]. Vid öppen styrning bestäms styrsignalen utan hänsyn till den faktiska utsignalen. Ett exempel på detta kan vara att en vattenkran öppnas ett förutbestämt antal grader, vilket ger ett visst vattenflöde. Vid öppen styrning tar man alltså inte hänsyn till eventuella avvikelser från modellen. Vid återkopplad styrning mäter man istället utsignalen och reglerar styrsignalen så att utsignalen matchar referenssignalen. I exemplet med vattenkranen skulle detta motsvara att man mäter vattenflödet och öppnar kranen olika mycket beroende på det faktiska flödet. Blir flödet för stort så stänger man exempelvis kranen lite.

2.5.1 PID-regulatorn

Vid återkopplad styrning är det vanligt att man använder sig av någon form av regulator som har till uppgift att beräkna en lämplig styrsignal baserad på felet mellan börvärde och ärvärde (uppmätt värde), vilket kallas reglerfel [31]. En av de vanligaste regulatorerna är PID-regulatorn som huvudsakligen består av tre delar, nämligen den proportionella, den integrerande och den deriverande delen.

Den proportionella delen tar reglerfelet och multiplicerar det med en parameter K_P. Om reglerfelet är 5 och $K_P = 2$ så bidrar P-delen med värdet 10 till styrsignalen. Den integrerande delen integrerar reglerfelet och multiplicerar det sedan med en parameter K_I. Den deriverande delen deriverar reglerfelet och multiplicerar det sedan med en parameter K_D. Slutligen summeras de olika delarna till en resulterande styrsignal.

Ett problem som kan uppstå vid implementeringen av PID-regulatorn är att integraldelen kan växa obegränsat. Detta blir problematiskt om systemet har fysiska begränsningar [32]. Till exempel kan inte höjdrodret vinklas mer än 18 grader, även om styrsignalen från regulatorn blir större än så. När reglerfelet börjar minska igen kommer styrsignalen minska, men höjdrodret kommer inte att röra sig före det att styrsignalen går under 18 grader. Ett sätt att förhindra detta är att begränsa hur stor integraldelen och styrsignalen tillåts att bli [32].

3

Hårdvara och konfiguration

För att kunna flyga autonomt kommer drönaren behöva använda sig av olika sensorer för att informera sig om sitt tillstånd och därefter reglera sig själv och avgöra vart den bör åka för att nå sitt mål. Detta avsnitt behandlar grunduppsättningen av hårdvara och hur denna konfigurerats. Som en del av konfigurationen motiveras valet av att använda ett kalmanfilter för positionsinläsning, varpå processen för framtagning av detta redovisas. Kalmanfiltret implementerades aldrig i slutprodukten, vilket diskuteras i avsnitt 7.2.2.

3.1 Komponenter

Eftersom projektet har fokus på mjukvaruutveckling användes ett modellflygplan i byggsats istället för att utforma och konstruera ett. Modellflygplanet som valdes är *Volantex Ranger Pusher Glider 2000 757-8*, främst för att det har långa och smala vingar vilket minskar det inducerade luftmotståndet och ökar den totala lyftkraften. Förutom detta har modellplanet en relativt rymlig flygarkabin vilket förväntades behövas för att få plats med all elektronik.

För att mäta drönarens status och styra systemet behövdes olika specifika komponenter väljas ut och integreras. Huvudsakligen behövdes sensorer för att mäta höjd, attityd, hastighet och position. Här följer en lista med de väsentligaste delkomponenterna som använts i systemet, med en förklaring av deras funktion och motivering till att de valts:

- **Bmp280 -** En barometer, vilken är en sorts tryckmätare, som används för att drönaren ska känna till sin höjd. Det uppmätta trycket jämförs mot ett referenstryck, varpå höjden över referenstrycket kan beräknas.
- **Bno055 -** Ett chip som innehåller en gyrometer, accelerometer och magnometer. Chipet har en inbyggd sensorfusion som ger ut attityd, linjär acceleration och andra värden.
- **Neo-6m -** En GPS-mottagare som ger ut nuvarande position i latitud och longitud.
- **Pca9685 -** Ett chip som kan skicka pwm-signaler på 16 olika portar. Detta chip används för att styra motorn och de servos som påverkar drönarens roderytor.
- **Raspberry Pi 3B -** Huvuddatorn i drönaren där alla beräkningar och logik för reglering, navigation och telemetri sker. Denna enkortsdator har ett operativsystem och är utrustad med generella in- och utdatapinnar för kommunikation med diverse sensorer och komponenter.
- **Teensy 3.2 -** En kompletterande mikrokontroller i drönaren. Den används för att läsa av från ett flertal sensorer, då Raspberry Pi inte har det antal kommunikationsmedier som krävs för att läsa av alla sensorer. Endast inläsning av sensordata

och konverteringsberäkningar sker på Teensy. Dessa sensorvärden skickas över till Raspberry Pi via USB.

- **FrSky x8r -** En radiokontroller-mottagare som används för att kunna styra drönaren manuellt, samt för att styra parametrar i drönarens mjukvara under flygning.
- **mpxv7002 -** En tryckskillnadsmätare som används för att beräkna drönarens hastighet i luften. Den består av två barometrar och ger ut skillnaden mellan den stillastående luftens tryck och trycket av luften i rörelse.

3.2 Anslutning och konfiguration av sensorer

Som tidigare nämnt sker alla beräkningar för den autonoma styrningen på huvuddatorn (Raspberry Pi). Däremot har ansvaret för sensorinläsning delats upp mellan huvuddatorn och den kompletterande mikrokontrollern, vilket visualiseras i bilaga A. I idealfallet skulle huvuddatorn läsa direkt från alla sensorer, då det minskar elektronikens komplexitet att behålla all kommunikation på samma enhet. På grund av huvuddatorns brist på tillgängliga kommunikationsportar har den kompletterande mikrokontrollen ansetts nödvändig för att läsa in viss sensordata. Vissa tillverkare erbjöd mjukvara för den kompletterande mikrokontrollen för vissa sensorer, vilket underlättade integreringen av dessa sensorer.

Många av sensormodulerna kunde konfigureras då deras sensor har en inbyggd mikrokontroller. Konfigureringsmässigt uppstod problem med GPS-modulen. Denna modul var ursprungligen konfigurerad för att ge positionsuppdateringar med en frekvens på 1 Hz, vilket kunde orsaka tvära svängningar när drönaren fick nya positionsvärden med för stora intervall. Flygning var möjligt med dessa inställningar, men en högre frekvens på positionsuppdateringarna ansågs nödvändigt för att göra navigationen smidigare.

För att uppnå detta designades ett Kalmanfilter som estimerade drönarens position genom att kombinera data från GPS-modulen och accelerometern för att få en högre uppdateringsfrekvens för positionsuppdatering. Hur detta implementerades beskrivs i avsnitt 3.3.

Ett Kalmanfilter med flera sensorer förväntades också ge bättre noggrannhet vid positionsinläsningen än endast GPS. I projektets slutskede upptäcktes dock ett sätt att läsa in GPS-data med 10 Hz vilket fungerade tillfredsställande, men arbetet med Kalmanfiltret fortsatte eftersom det fanns potential för bättre positionsestimering. Dock integrerades inte Kalmanfiltret med drönarens styrsystem på grund av tidsbrist.

3.3 Sensorfusion med Kalmanfilter för en bättre uppskattning av positionen

Som behandlats i avsnitt 2.4 kan man få så kallat brus och osäkerhet i mätningarna av olika anledningar. Detta är något GPS-mottagaren visade sig ha. Det var även önskvärt att ta in tätare uppdateringar av positionen än de som kunde ges av GPS-mottagaren. (Detta beskrivs mer i avsnitt 3.2.) Därför valdes det att kombinera värden från GPS-mottagaren med värdena från accelerometern genom Kalmanfiltrering. Bakomliggande teori för detta behandlas i avsnitt 2.4.1.

3.3.1 Inställningar för Kalmanfiltret

Accelerometervärdena konverterades till kardinalriktningar enligt avsnitt 2.4.4 och GPS-positionerna konverterades till meter enligt avsnitt 2.4.5. För att få den magnetiska deklinationen för att kunna filtrera ut den från den aktuella positionen användes klassen `GeomagneticField` från Androids API [33].

För positionsestimering kan man sätta variablerna i avsnitt 2.4.1 till:

$\mathbf{x}$ är en vektor av position och hastighet $\begin{bmatrix}\mathbf{p}\\ \mathbf{v}\end{bmatrix}$

$\mathbf{F}$ är $\begin{bmatrix}1 & \Delta t\\ 0 & 1\end{bmatrix}$

$\mathbf{B}$ är $\begin{bmatrix}\frac{\Delta t^2}{2}\\ \Delta t\end{bmatrix}$

$\vec{\mathbf{u}}$ är accelerationen.

$\mathbf{P}$ kan initieras som $\mathbf{I}$ [24]

$\mathbf{H}$ kan ignoreras eftersom vi ser till att alla mätvärden har rätt enheter innan de skickas till Kalmanfiltret.

$\mathbf{Q}$ är kovariansmatrisen för bruset vid estimeringen av positionen via accelerometern.

$\mathbf{R}$ är kovariansmatrisen för bruset vid observationen av positionen via GPS-sensorn.

3.3.2 Beräkningar av kovariansmatriser för Kalmanfiltret

Inledningsvis gjordes försök att estimera standardavvikelsen för GPS-sensorn och accelerometern för att kunna beräkna $\mathbf{Q}$ och $\mathbf{R}$ enligt den första metoden beskriven i teorikapitlet under avsnitt 2.4.3.

Dessa beräknades genom att låta de två sensorerna ligga stilla och samla mätdata. Ett problem som uppkom var att accelerometerns medelacceleration i alla riktningar blev det i ekvation 3.1. Dock borde den vara cirka noll vid datainsamling över en längre tid. (Se figur 2.1 för axlarnas riktningar i förhållande till flygplanet.) Anledningen till detta kan vara bland annat små variationer i tillverkningsprocessen eller lagringsförhållanden [34]. (Mängden accelerometervärden insamlade här var 4812 stycken.)

$$\begin{matrix}x:\\ y:\\ z:\end{matrix}\begin{bmatrix}-0,0513674147963\\ 0,0151309226933\\ 0,0229821280133\end{bmatrix} \tag{3.1}$$

Den slutliga standardavvikelsen för accelerometervärdena beräknades till

$$0,0834478392961\ \mathrm{m\,s^{-2}} \tag{3.2}$$

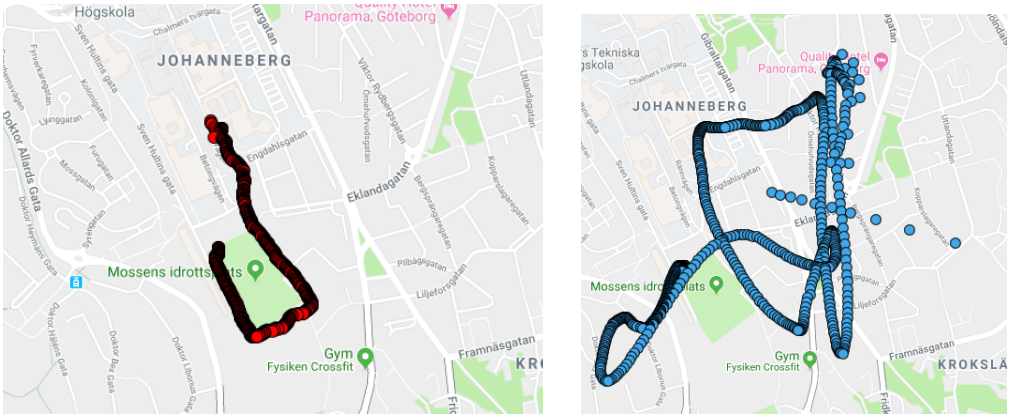
och för GPS-värdena till

$$56,6954833472\ \mathrm{m}.$$

Resultat av fysiska tester av Kalmanfiltret med dessa värden satta diagonalt i $\mathbf{Q}$ och $\mathbf{R}$ enligt första metoden beskriven i avsnitt 2.4.3, se ekvation 3.3. I detta fall hade GPS-sensorn en frekvens på 1 Hz och ett prediktionsvärde för varje korrektionsvärde enligt figur

2.4. Ett problem med detta var att accelerometern hade en högre uppdateringsfrekvens än GPS-sensorn vilket ledde till att man inte kunde kombinera ett accelerometervärde per GPS-värde. Detta löstes genom användning av medelvärdet från alla accelerometervärden sedan den senaste uppdateringen. Resultatet visas i figur 3.1. Testerna genomfördes genom att bära runt flygplanet och samla mätdata.

$$\mathbf{Q} = \begin{bmatrix} 0,0834478392961 & 0 \\ 0 & 0,0834478392961 \end{bmatrix} \mathbf{R} = \begin{bmatrix} 56,6954833472 & 0 \\ 0 & 56,6954833472 \end{bmatrix} \tag{3.3}$$



Figur 3.1: Jämförelse mellan positionsutläsning med ren GPS-data (röda värden) och Kalmanfiltrets estimering av positionen (blåa värden) med **Q** och **R** enligt ekvation 3.3. Bild skapad med copypastemap.com. Karta © 2019 Google.

Som syns i figur 3.1 gav testet undermåliga resultat med dessa inställningar. GPS-sensorn följer ensam den testade vägen bra medan den Kalmanfiltrerade positionsutläsningen visade till synes slumpmässiga värden. Varför dessa uppmätta värden av standardavvikelserna fungerade undermåligt diskuteras vidare i avsnitt 7.2.1 i diskussionskapitlet.

En annan metod för att bestämma **Q** och **R** är att helt enkelt gissa deras värden, som också nämns i avsnitt 2.4.3 i teorikapitlet. Efter det misslyckade testet med de uppmätta standardavvikelserna inmatade i **Q** och **R** testades därför att ge accelerometern och GPS:en samma standardavvikelse, enligt ekvation 3.4. Här sattes även GPS-sensorns frekvens till 10 Hz och flera estimeringar användes per observation enligt 2.5. Resultat av detta test visas i figur 3.2.

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.4}$$

Detta gav ett bra resultat. Dock, eftersom detta test utfördes genom att drönaren bars runt kan man anta att större problem skulle kunna uppkomma vid större accelerationer under flygning. Därmed sattes standardavvikelsen till 2 i **Q** och 0,1 i **R** (enligt ekvation 3.5). Resultatet av detta kan ses i figur 3.3. Detta diskuteras vidare i avsnitt 7.2.2.

$$\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 0,1 & 0 \\ 0 & 0,1 \end{bmatrix} \tag{3.5}$$

Figur 3.2: Jämförelse mellan GPS-data (röda värden) och Kalmanfiltrets estimering av positionen (blåa värden) med **Q** och **R** enligt ekvation 3.4. Notera att den Kalmanfiltrerade positionen ger fler värden än den från endast GPS-mätningar, men att dessa ibland avviker något (knappt synligt) från banan. Bild skapad med copypastemap.com. Karta © 2019 Google.

Figur 3.3: Jämförelse mellan GPS-data och Kalmanfiltrets estimering av positionen med **Q** och **R** enligt ekvation 3.5. Här syns en god noggrannhet hos den Kalmanfiltrerade positionen, som följer samma bana som den rena GPS-mätningen men ger fler värden. Bild skapad med copypastemap.com. Karta © 2019 Google.

Det är svårt att estimera position genom att dubbelintegrera accelerometern [35]. Detta visar resultaten i figur 3.1 och 3.2. Därav kan vi se i figur 3.3 att Kalmanfiltret fungerar bättre när man tar mer hänsyn till GPS:ens värden än accelerometerns vid nya observationer. Accelerometern används dock fortfarande för att estimera riktningsförändringar mellan observationer, vilket diskuteras i avsnitt 3.3.3.

3.3.3 Simulering av data för Kalmanfiltret

För att få mer data som kunde verifiera att Kalmanfiltret fungerade korrekt skapades ett system för att generera falsk sensordata.

Testen utfördes genom att först mata in två startkoordinater till Kalmanfiltret. Den andra koordinaten sattes ut en meter i den nordliga riktningen från den första, en sekund senare. Därmed satte Kalmanfiltret en starthastighet på 1 m/s norr. Efter detta applicerades olika accelerationer och, för de tre första figurerna, beräknades nästkommande koordinat manuellt till samma position dit estimationen hade tagit den.

Vid de två senare testerna gavs observationen och prediktionen olika resultat för att testa inflytan av $\mathbf{Q}$ och $\mathbf{R}$.

De första tre testerna skedde med $\mathbf{Q}$ och $\mathbf{R}$ inställda enligt ekvation 3.5, och de två sista med dem enligt ekvation 3.6 respektive 3.7 (med undantag av testerna för figur 3.4d och 3.4e där prediktionen och observationen skiljer sig).

Inledningsvis testades konstant acceleration norrut genom att sätta accelerationen till noll i alla riktningar. Resultatet av detta illustreras i figur 3.4a och simulationsdatan finns i simuleringsdata G.1. Därefter testades att applicera en acceleration på 1m/s^2 österut efter den andra observationen. Resultatet av detta finns i figur 3.4b och simulationsdatan finns i bilaga G.2. Resultatet av dessa tester tyder på god funktionalitet hos estimeringen mellan observationer vid perfekta accelerometervärden.

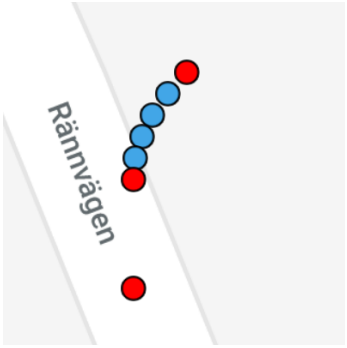
Därefter upprepades samma test, men Kalmanfiltret tilläts estimera fler positioner efter den första sekvensen. Resultatet av detta finns i figur 3.4c och simulationsdatan finns i bilaga G.3. Här kan vi se att på grund av lågt förtroende för accelerometern sätts den estimerade riktningen efter den tredje observationen till största del baserat på de två tidigare observationerna från GPS:en. Därmed fortsatte banan med en vinkel som inte riktigt var väntad, rakt ut från den sista observationen.

De sista två testerna undersökte sammanfogning då observation och estimation skilde sig. I dessa fall sattes den sista observationen en meter norr om den andra medan accelerationen sattes till 1 m/s^2 öst därefter. I figur 3.4d sattes $\mathbf{Q}$ och $\mathbf{R}$ enligt ekvation 3.6 och i figur 3.4e sattes de enligt ekvation 3.7. Här kan vi tydligt se att simulationen där högre tillit sattes till observationen fick en bättre slutlig estimering jämfört med simulationen med lägre tillit. Simulationsdatan för dessa tester finns i bilaga G.4.

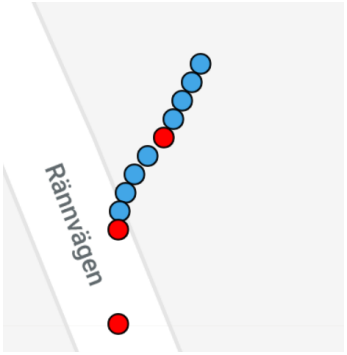
$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \tag{3.6}$$

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{R} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \tag{3.7}$$

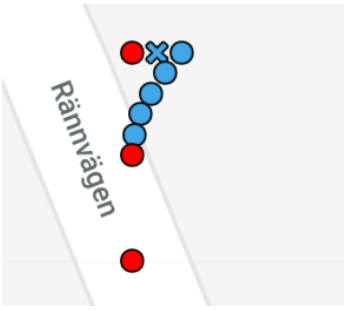
(a) Kalmanfilter vid konstant hastighet 1 m/s norr och ingen acceleration.



(b) Kalmanfilter vid initialhastighet 1 m/s norr och acceleration med 1 m/s^2 öst efter den andra observationen.



(c) Kalmanfilter vid initialhastighet 1 m/s norr, acceleration med 1 m/s^2 öst efter den andra observationen och ingen acceleration efter den tredje observationen.



(d) Kalmanfilter vid initialhastighet 1 m/s norr, acceleration med 1 m/s^2 öst efter den andra observationen. Här hamnar dock estimeringen och observationen vid olika positioner så Kalmanfiltret gör en estimation med **Q** och **R** enligt ekvation 3.6.

(e) Kalmanfilter vid initialhastighet 1 m/s norr, acceleration med 1 m/s^2 öst efter den andra observationen. Här hamnar dock estimeringen och observationen vid olika positioner så Kalmanfiltret gör den slutliga estimationen med **Q** och **R** enligt ekvation 3.7.

Figur 3.4: Resultat av att Kalmanfiltrera positionsinläsning vid simulering med falska sensorvärden. De röda prickarna representerar positionen vid observation av nytt värde medan de blåa representerar Kalmanfiltrets prediktion. Se bilaga G för värden använda vid simuleringen. Bild skapad med copypastemap.com. Karta © 2019 Google.

4

Systemmodellering och reglerdesign

För att få flygplanet att navigera och flyga autonomt behövdes ett komplett reglersystem, som med hjälp av sensorer läser in flygplanets tillstånd och sedan styr flygplanet med roder och motorkraft. I det här kapitlet beskrivs arbetet med att ta fram ett sådant system. I de två första avsnitten redogörs för hur modellerna för aerodynamik och motor har tagits fram. Sedan skildras processen där dessa modeller importerades till mjukvaran Simulink för att designa reglersystemet och bestämma dess parametrar.

4.1 Systemmodellering av aerodynamik

För att på ett effektivt sätt kunna designa ett reglersystem behövdes matematiska modeller över systemen som skulle regleras. I tidigare studier som gruppen studerat har framför allt tre olika metoder för modellering av flygande farkoster använts; *analytisk*, *black box*- samt *grey box* modeller (svenska *svart låda* respektive *grå låda*) [36][37]. Även i svenska publikationer är det vanligt att de engelska uttrycken används, vilket även görs fortsättningsvis i den här rapporten.

I den analytiska metoden bestäms aerodynamiska parametrar genom antingen vindtunneltester eller simuleringar. Parametrarna beskriver flygplanets beteende under flygning och kan till exempel specificera hur stort moment som skapas kring en specifik axel när planet lutar ett visst antal grader i någon riktning.

Vid black box modellering används istället mjukvara för att utifrån det fysiska systemets uppmätta in- och utsignaler hitta en bakomliggande modell [38]. Ett exempel på utsignal kan vara flygplanets pitch, som i sin tur påverkas av höjdrodrets vinkel vilken kan ses som en insignal. Ett problem med denna typ av modell är att det generellt sett är svårt att sätta sig in i hur dynamiken fungerar, då sambanden mellan parametrarna i modellen ofta blir komplexa, därav namnet svart låda.

Den sista modellen, grey box, är en hybrid mellan de två tidigare metoderna. Modellen beskrivs med analytiskt framtagna ekvationer, men man låter sedan mjukvara bestämma parametrarna i modellen så att modellen matchar uppmätt in- och utdata [38]. På detta sätt kan man få en databaserad modell, där parametrarna fortfarande går att jämföra med helt analytiska beräkningar.

Studier genomförda vid Embry-Riddle Aeronautical University visade att grey box och black box modellerna gav något bättre resultat än den helt analytiska när de testades på träningsdata [37]. Med träningsdata menas här den bakomliggande data som använts för att ta fram modellen. När modellerna istället jämfördes med ny data så hade dock de

två modellerna en tendens att uppvisa ungefär samma resultat som den helt analytiska modellen.

För aerodynamikmodelleringarna valdes den analytiska metod som använder sig av datorsimuleringar. Valet motiveras framför allt med att den här typen av modell är mer flexibel vid förändringar. Om till exempel flygplanets vikt och tyngdpunktsläge förändras räcker det med att utföra nya simuleringar. Hade istället någon av de två andra metoderna använts hade man behövt genomföra nya flygningar för att samla in data. Den analytiska metoden valdes även på grund av att den kan anses mer transparent, med intuitiva parametrar som bidrar till ökad systemförståelse.

Som simuleringsmjukvara för aerodynamikmodellerna valde gruppen att använda sig av XFLR5. Programmet har öppen källkod och kan användas för att bestämma aerodynamiska egenskaper hos flygplan. Mjukvaran kan dessutom ta fram två linjäriserade tillståndsmodeller, en longitudinell som beskriver flygplanets dynamik i höjdled, samt en lateral som beskriver dynamiken i horisontalplanet. Tester utförda vid NASA har bland annat visat att programmet är väl lämpat vid modellering av mindre flygplan vid relativt låga hastigheter, vilket är huvudanledningen till att mjukvaran valdes [39].

4.1.1 Flygplanets grunddata

Modelleringen av flygplanet krävde att dess fysiska egenskaper, så som storlek, utformning och vikt, bestämdes.

En våg användes för att mäta den totala vikten på flygplanet med all hårdvara installerad. Även större komponenter så som batterier, motor, vingar och enkortsdator vägdes separat eftersom dessa uppgifter behövdes för att XFLR5 skulle kunna uppskatta flygplanets trögheтsmoment kring dess masscentrum. Placeringen av dessa komponenters masscentrum i flygplanet mättes med linjal.

Nedan följer en tabell över flygplanets grunddata som bland annat legat till grund för modelleringen i XFLR5.

Vingspann	200 cm
Längd	108,5 cm
Medelkorda	18,7 cm
Vingyta	3690 cm^2
Total vikt	2,158 kg

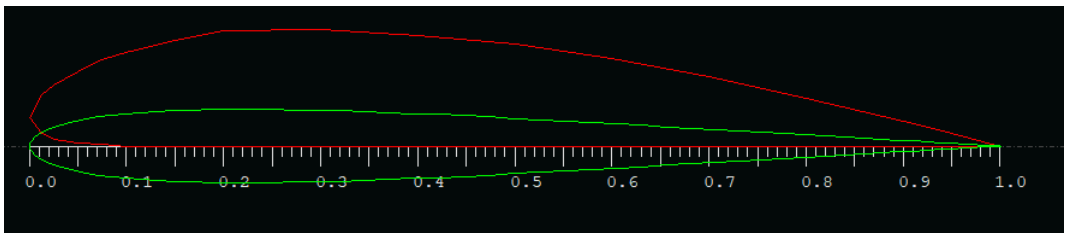
Tabell 4.1: Flygplanets fysiska specifikationer.

4.1.2 Identifiering av vingprofiler

För att modelleringen skulle stämma väl behövdes även utformningen på de olika vingprofilerna på flygplanet bestämmas. Detta gjordes genom att först mäta specifika mått på vingprofilen och sedan jämföra dessa mot en databas med standardiserade vingprofiler för att hitta en som var lik, vars data sedan kunde användas i modellerna.

Vingprofilernas längd brukar betecknas *korda* och är längden på den raka linje som går från vingens framkant till dess bakkant [40]. Vingprofilers tjocklek anges som kvoten mellan kordans längd och den maximala tjockleken, uttryckt i procent [40].

Huvudvingen på det valda flygplanet hade en platt undersida och maximal tjocklek vid ungefär 30% av dess totala längd. Vid vingroten uppmättes den maximala tjockleken till 23mm och kordan till 200 mm, vilket gav 11,5% tjocklek. Den vingprofilen som bäst passade dessa mått bedömdes vara en som kallas *Rhode St. Genese 32* vilken har en tjocklek på 11,9% [41]. På samma sätt mättes vingprofilerna för den vertikala och horisontella stabilisatorn på flyplanets bakdel. Dessa profiler var istället symmetriska och hade en tjocklek på ungefär 7%, vilket visade sig stämma väl överens med profilen *S9033* som har en tjocklek på 7,5% [42].



Figur 4.1: Bild från simuleringsmjukvaran XFLR5 med de två vingprofilerna. Huvudvingen representeras här i rött och stabilisatorns ytor i grönt.

4.1.3 Framtagning av linjäriserade tillståndsmodeller

När hela flygplanet var modellerat kunde XFLR5 beräkna olika aerodynamiska koefficienter samt ta fram de två tidigare nämnda tillståndsmodellerna, som behövdes för reglerdesignen. Tillståndsmodellerna beräknades av XFLR5 kring en linjäriserad arbetspunkt, som motsvarar att planet håller konstant anfallsvinkel, höjd, kurs, rodervinklar och hastighet genom luften. Modellerna blir sedan giltiga för relativt små förändringar kring arbetspunkten. Den av XFLR5 beräknade arbetspunkten presenteras i tabell 4.2 nedan.

Anfallsvinkel	-0,60°
Hastighet	15,78 m/s
Höjdroder	0,0°
Sidoroder	0,0°
Skevroder	0,0°

Tabell 4.2: Beräknad arbetspunkt enligt XFLR5.

Tidiga testflygningar visade att hastigheten, beräknad utifrån GPS-data, vid uttrimmad flygning var cirka 15 m/s, vilket stämmer bra med den beräknade arbetspunkten. Med uttrimmad menas i det här fallet att rodrens utgångslägen är satta så att när kontrollerna släpps vid manuell flygning bibehålls flygplanets höjd och kurs. Att anfallsvinkeln blir något negativ (det vill säga att nosen pekar något under horisonten) är väntat och kan förklaras med att vingprofilens utformning gör att vingen bidrar med positiv lyftkraft ända ner till cirka -4,0° anfallsvinkel.

Efter linjäriseringen kring arbetspunkten genererade mjukvaran bland annat de två tillståndsmodeller som senare ligger till grund för reglerdesignen. Dessa två tillståndsmodeller återfinns i sin helhet i bilaga F.

4.1.4 Servomodell

De tidigare framtagna aerodynamiska modellerna tar in rodrens faktiska lägen. Detta innebär att rodervinklarna förutsätts sättas till önskade värden utan någon fördröjning i simuleringen. I verkligheten kommer det att finnas en fördröjning från det att programmet begär en rodervinkel tills dess att servon har satt denna vinkel, och därför behövdes en modell som beskriver denna fördröjning.

Enligt en norsk studie från år 2007 kan enklare elservon för hobbyprodukter relativt väl approximeras med hjälp av en begränsning av rotationshastigheten [36].

Elservon som används till rodren på projektets flygplan är av typen *Turnigy TG9d*. Enligt leverantörens specifikationer är rotationshastigheten 0,10 sekunder per 60 grader vid 4,8 volt och 0,09 sekunder per 60 grader vid 6,0 volt [43]. För att få rotationshastigheten som gäller vid 5,0 volt (vilket används i projektet) interpoleras rotationshastigheterna. Slutligen erhålls en rotationshastighet på 590 grader/sekund.

I simulinkmodellen implementeras sedan detta med ett *rate-limiter-block*. Blocket fungerar på så sätt att det begränsar den maximala förändringshastigheten, i detta fall enligt uträkningen till max 590 grader/sekund.

4.2 Systemmodellering av motordynamik

Förutom den aerodynamiska modellen behövdes även en modell över sambandet mellan begärd motoreffekt och flygplanets hastighet genom luften. För att modellera motordynamiken valdes black box metoden, som beskrivs i början av avsnitt 4.1. Till skillnad mot aerodynamikmodellen så förväntas inte motordynamiken att förändras nämnvärt under arbetets gång. Behovet av en flexibel modell var alltså inte lika stort här. Istället fokuserades på att få en så rättvisande modell som möjligt. Förhoppningen var även att en black box modell skulle gå snabbare att ta fram jämfört med den helt analytiska.

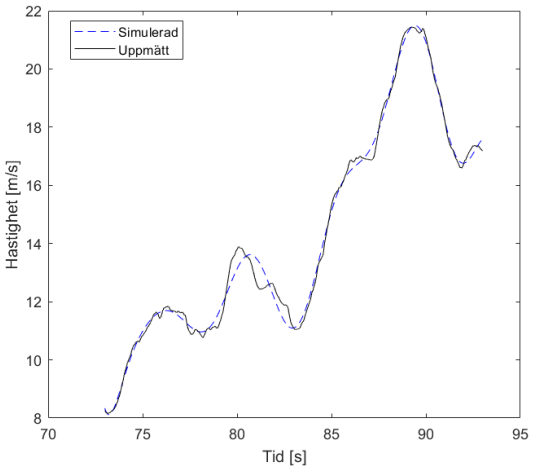
För att samla data genomfördes en testflygning där flygplanets hastighet genom luften samt begärd effekt från motorn mättes. Mätdatan samlades med 100 Hz, alltså 100 gånger per sekund. Därefter användes Matlabs *System Identification Toolbox* för att hitta en lämplig överföringsfunktion mellan begärd motoreffekt och hastighet.

Två serier med mätdata från testflygningen valdes ut. Den första mätserien användes för att ta fram approximationen och den andra mätserien användes för att validera den framtagna modellen. Mätserierna valdes med urvalskriteriet att relativt stora hastighetsvariationer förekom och förändringar i pitch- och rollvinkel var relativt låga. Förändringar i pitch- och rollvinkel påverkar luftmotståndet och på så sätt även hastigheten, vilket var anledningen till att mätserier med låga variationer på dessa eftersöktes.

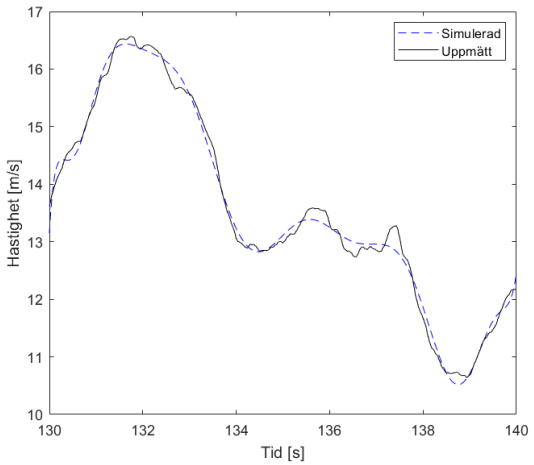
Empiriska tester visade att den överföringsfunktion som passade bäst hade sex poler och ett nollställe. Den framtagna överföringsfunktionen återfinns i bilaga F.

I figur 4.2 och 4.3 nedan presenteras den uppmätta och simulerade hastigheten på den första respektive andra mätserien.

De båda mätserierna gav enligt Matlab en passning på 87,15% respektive 90,66%, vilket av gruppen anses vara relativt bra approximationer.



Figur 4.2: Första mätserien med uppmätt och simulerad hastighet.



Figur 4.3: Andra mätserien med uppmätt och simulerad hastighet.

4.3 Design av reglersystem

Flera tidigare studier har visat goda resultat vid användande av PID-regulatorer för att styra mindre flygplans roll och pitch [36][44]. Det finns ett flertal mer specifika lösningar på regulatorer som kan antas ge något bättre prestanda, men gruppen har ändå valt att använda sig av PID-regulatorer. En anledning till detta är att medlemmarna i projektgruppen främst har erfarenhet av dessa, men också att PID-regulatorer är väl beprövade inom området.

En väl beprövad variant på regulatorstruktur för mindre flygplan använder sig av kaskadreglering [44]. Grundidén är att låta en inre regulator reglera den snabbaste dynamiken och en yttre reglera den långsammare. Exempelvis kan den inre regulatorn reglera pitch och den yttre reglera höjd, där pitch har en snabbare dynamik än höjden. Den yttre loopen reglerar således höjden genom att bestämma börvärden till den inre loopen som i sin tur bestämmer höjdrodrets vinkel. Ett exempel på detta kan ses i figur 4.4.

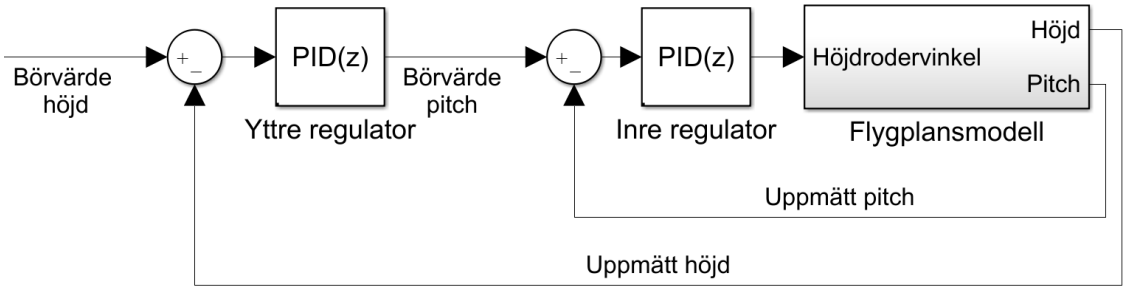
Gruppen valde att tillämpa den ovannämnda strukturen för både reglering av höjd och kurs, vilket beskrivs mer i detalj i de två efterkommande avsnitten.

4.3.1 Höjdreglering

För höjdreglering användes den regulatorstruktur som beskrivs i figur 4.4. Regulatorn tar alltså in en önskad höjd och styr sedan höjdrodrets vinkel för att uppnå denna.

Regulatorstrukturen byggdes upp i Simulink tillsammans med den tidigare framtagna longitudinella modellen. PID-regulatorernas utsignaler begränsades, exempelvis begränsades utsignalen från den innersta loopen till +/- 18° eftersom detta är största respektive minsta rodervinklar som fysiskt är möjliga att ge med den givna hårdvaran.

Den yttre regulatorns utsignal begränsades till +/- 20°, vilket då blir den största respektive minsta pitch som kan ges. Tidigare genomförda flygtester visade att man under normal lugn flygning inte överstiger dessa värden. En annan anledning att begränsa pitchvärdena var att den framtagna modellen stämmer bäst överens för relativt små vinkelförändringar. För riktigt stora pitchvinklar stämmer alltså inte modellen och då vet man inte hur planet



Figur 4.4: Regulatorstruktur för höjdreglering.

kommer att bete sig.

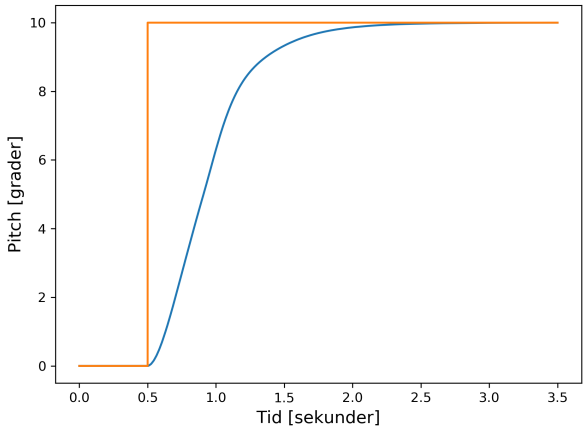
När regulatorstrukturen var klar användes en funktion i Matlab som heter *PID Tuner App* för att hitta lämpliga parametrar för den inre regulatorn. För att utvärdera parametervärdena studerades stegsvaret för enbart den inre loopen när börvärdet (begärd pitch) sattes till tio grader uppåt.

Vanligtvis innefattar reglerdesign en avvägning mellan hur snabbt systemet ska reagera, samt hur stabilt det blir [31]. Med detta i åtanke valde gruppen vid framtagningen av PID-parametrar att lägga tyngd vid ett stabilt system, snarare än ett snabbt. Ett begrepp som används för att beskriva ett systems snabbhet är *stigtid*. Stigtid definieras som ”tiden det tar för utsignalen y(t) att gå från 10% till 90% av dess slutvärde” [31].

Vid optimeringen av den inre regulatorn sattes ett önskemål om en ungefärlig stigtid på 0,5 sekunder. Det önskade beteendet uppnåddes med en PD-regulator med parametrar och stegsvar enligt tabell 4.3 och figur 4.5.

PID	Kp	Ki	Kd	N	Max	Min
Inre	1,01	0	0,1	190	18	-18

Tabell 4.3: Parametrar för den inre PID-regulatorn, som reglerar flygplanets pitch.



Figur 4.5: Stegsvar för pitch med börvärde 10 grader.

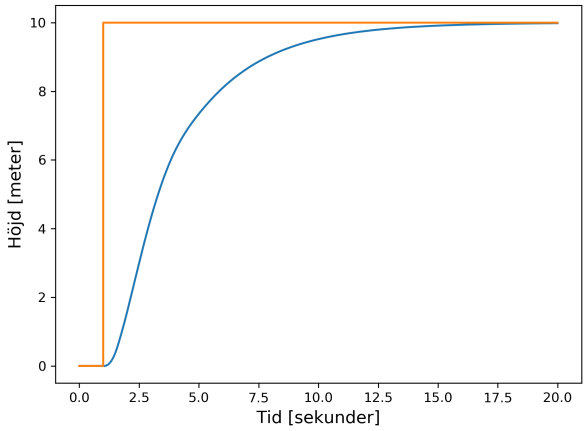
Som kan utläsas ur figur 4.5 blir stigtiden för pitchen cirka 0.6 sekunder utan kvarstående fel.

Efter att parametrarna för den inre regulatorn tagits fram kopplades återigen den yttre loopen in i simuleringen och hela systemets stegsvar för olika parametrar studerades med hjälp av PID Tuner App. Stegsvaret för en stigning från noll till tio meter studerades. Här sattes önskemål om en längre stigtid, då dynamiken i höjdled är långsammare. Efter erfarenheter från flygtestet sattes önskemålet till sex sekunder.

Det önskade beteendet uppnåddes med en enkel P-regulator med parametrar och stegsvar enligt tabell 4.4 och figur 4.6.

PID	Kp	Ki	Kd	N	Max	Min
Yttre	1,2	0	0	0	20	20

Tabell 4.4: Parametrar för den yttre PID-regulatorn, som reglerar flygplanets höjd.



Figur 4.6: Stegsvar för den kompletta höjdregleringen med börvärde 10 meter.

Som kan utläsas ur figur 4.6 erhålls en maximal stighastighet för flygplanet på cirka två m/s och en stigtid för systemet på cirka sex sekunder.

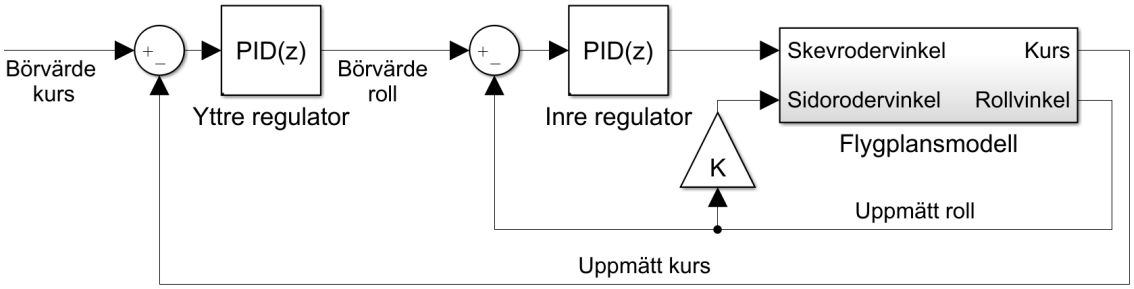
4.3.2 Kursreglering

För kursregleringen i lateralplanet valdes en struktur med samma grundupplägg som för höjdregleringen. Den yttre regulatorn tar in kursfelet och matar ut börvärde för rollvinkel. Den inre regulatorn tar in felet i rollvinkel och matar ut börvärde för skevrodren.

För kursregleringen spelar även sidorodret roll. När skevroder ansätts så att flygplanet till exempel lutar åt vänster för att svänga åt vänster ger detta samtidigt upphov till en svag gir åt höger. Anledningen till detta är att lyftkraften på höger vinge ökar, vilket också gör att luftmotståndet på den högra vingen ökar och planet vill vrida sig åt höger. Flygplanet kommer fortfarande att svänga åt vänster, men det kommer att driva i sidled, vilket ger ökat luftmotstånd. Sidorodret kan användas för att motverka denna gir och på så sätt ge en sväng där planet inte driver i sidled.

Det finns flera olika metoder för att mäta hur mycket planet kanar in i svängen. Ingen av dessa ansågs dock möjliga att tillämpa inom ramen för projektet. Därför valdes en enklare öppen styrning för sidorodret. Detta innebär att den uppmätta rollvinkeln multipliceras med en konstant, som då ger hur mycket sidoroder som ska ansättas. Initialt har denna konstant satts till 0,1, vilket innebär att 30 grader rollvinkel resulterar i tre

grader sidoroder. Denna konstant kan justeras allt eftersom mer data från testflygningar blir tillgängliga. I figur 4.7 syns strukturen för kursregulatorn med den öppna styrningen som en förstärkning märkt med bokstaven K.

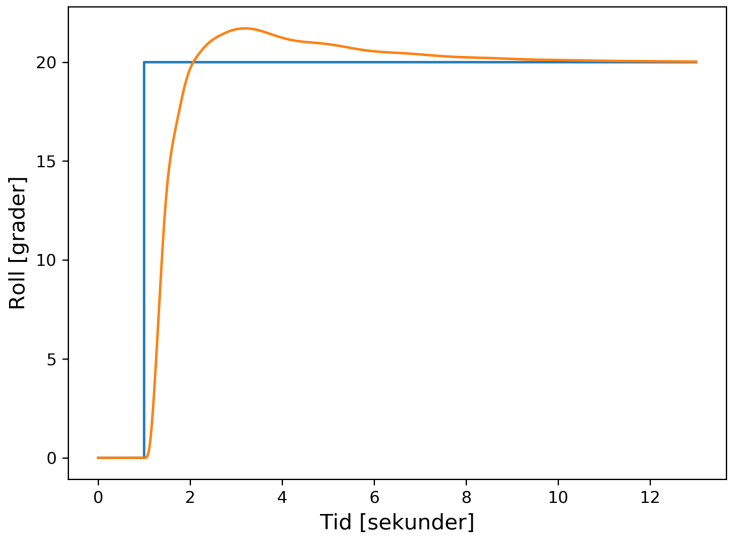


Figur 4.7: Den valda regulatorstrukturen för reglering av kurs.

Parametrarna för regulatorerna bestämdes även här med hjälp av PID Tuner App i Matlab. Först simulerades endast den inre loopen och stegsvaret för 20 grader roll studerades. Önskemålet på stigtid sattes till en sekund. Här visade det sig dock svårt att undvika översläng, alltså att rollvinkeln översteg den önskade med några procent. De parametrar som gav det bästa simulerade utfallet presenteras nedan i tabell 4.5 och dess stegsvar syns i figur 4.8.

PID	Kp	Ki	Kd	N	Max	Min
Inre	-1,8	-0,6	-0,3	17	18	18

Tabell 4.5: Parametrar för den inre PID-regulatorn, som reglerar flygplanets rollvinkel.



Figur 4.8: Stegsvar för roll med börvärde 20 grader.

Som kan ses i figur 4.8 blir stigtiden för systemet ungefär 1,2 sekunder, med cirka 8% översläng.

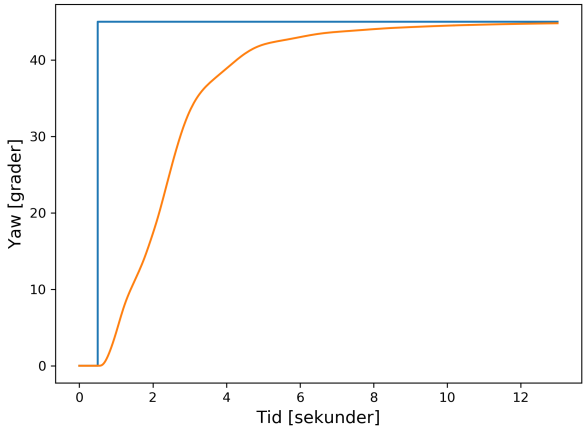
Med den inre regulatorns parametrar på plats kopplades återigen den yttre loopen in till systemet och PID Tuner App användes för att hitta lämpliga parametrar för den yttre

regulatorn. Önskemålet på stigtid sattes här till tre sekunder.

Stegsvaret för en sväng från en rakt nordlig kurs (noll grader) till nordost (45 grader) studerades. Lämpliga parametrar och det resulterande stegsvaret presenteras i tabell 4.6 och figur 4.9.

PID	Kp	Ki	Kd	N	Max	Min
Yttre	0,95	0	0	0	20	20

Tabell 4.6: Parametrar för den yttre PID-regulatorn, som reglerar flygplanets kurs.



Figur 4.9: Stegsvar för den kompletta laterala navigationen med börvärde 45 grader.

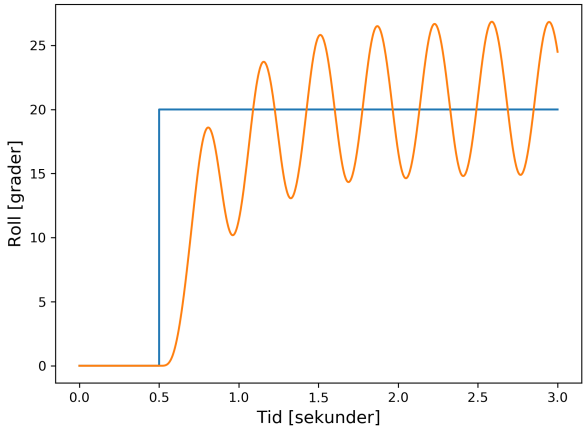
Som kan utläsas från figur 4.9 uppgår den maximala svänghastigheten till cirka 15 grader per sekund och stigtiden för systemet blir cirka 3,5 sekunder.

Vid det första flygtestet med dessa parametrar observerades relativt kraftiga oscillationer i roll-led. Planet rörde sig fram och tillbaka i roll-led med cirka tre svängningar per sekund. En trend att oscillationerna blev starkare ju snabbare planet flög kunde observeras. GPS-data från testflygningen visade att flygplanet som snabbast färdats i cirka 20 m/s, vilket är 5 m/s snabbare än vad modellen är linjäriserad kring. Med anledning av detta genomfördes nya simuleringar i XFLR5 med 50% högre hastighet (alltså 22,5 m/s) och den nya tillståndsmodellen importerades sedan till modellen i Simulink. I figur 4.10 nedan syns resultatet från en simulering med endast den inre loopen där börvärdet satts till 30 grader. Detta motsvarar alltså att man begär att planet ska gå från noll till 30 graders rollvinkel.

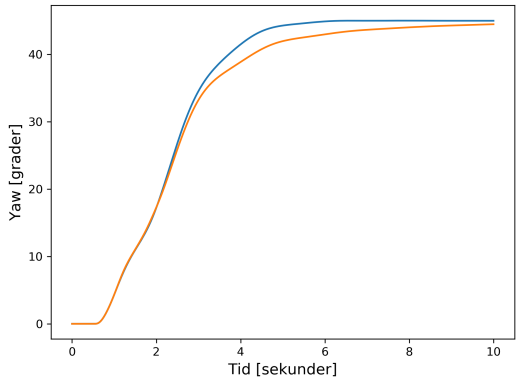
Precis som under testflygningen uppträder här starka oscillationer. På grund av detta utarbetades nya parametrar som gav tillfredställande stegsvar vid både låga *och* höga hastigheter.

PID	Kp	Ki	Kd	N	Max	Min
Inre	-1,0	0	-0,05	17	18	18
Yttre	1,0	0	0	0	20	20

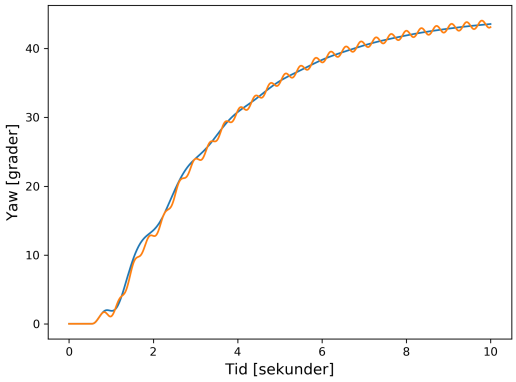
Tabell 4.7: Nya parametrar för kursreglering som fungerar bra också vid högre hastigheter.



Figur 4.10: Stegsvar för roll med 50% högre hastighet.



Figur 4.11: Stegsvar för yaw vid låg hastighet. Gamla parametrar i rött och nya i blått.



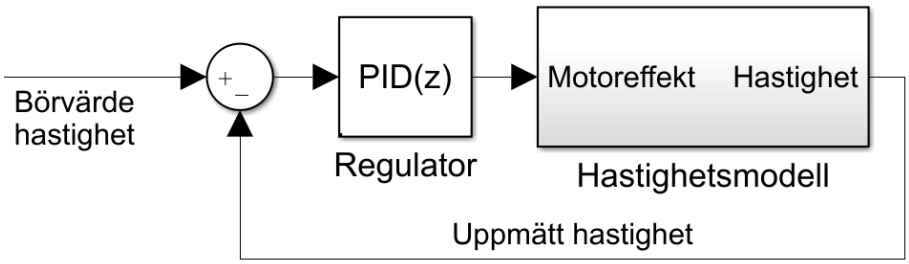
Figur 4.12: Stegsvar för yaw vid hög hastighet. Gamla parametrar i rött och nya i blått.

I figur 4.11 och 4.12 redovisas de simulerade stegsvaren vid hastighet 15 respektive 22,5 m/s. I båda fallen med begärd yaw på 45 grader. De röda kurvorna visar stegsvaret med de gamla regulatorparametrarna och de blåa med de nya. Som man kan se i figur 4.12 så uppstår endast lättare oscillationer vid höga hastigheter med de nya parametrarna. Under senare testflygningar verifierades också att de nya parametrarna fungerade bättre, nu även vid högre hastigheter.

4.3.3 Hastighetsreglering

Hastighetsregleringen sköts av en ensam PID-regulator enligt figur 4.13.

Som nämns i föregående avsnitt 4.3.2, så uppstod oscillationer i roll med den första framtagna parameteruppsättningen för kursregleringen. För att beräkna parametrarna till hastighetsregulatorn valde gruppen därför istället att använda sig av applikationen *Control System Tuner* i mjukvaran Simulink. Anledningen till detta är att man i Control System Tuner enklare kan ta hänsyn till fler parametrar, till exempel krav på stabilitet. Control System Tuner fungerar på så sätt att man anger vilka parametrar eller regulatorer som ska optimeras, samt olika mål man vill uppnå. Målen kan definieras på många olika sätt, men



Figur 4.13: Den valda strukturen för hastighetsreglering.

gruppen har valt att specificera gränser för stabilitetsmarginaler, referensföljning samt maximal översläng.

Stabilitetsmarginalerna beskrivs med två olika parametrar, amplitud- och fasmarginal. Amplitud- och fasmarginal beskriver hur mycket förstärkningen respektive fasen för det öppna systemet kan förändras innan systemet blir instabilt [31].

Att bestämma exakt hur stor fas- och amplitudmarginalen bör vara är svårt, då detta varierar mellan olika system. Därför sätts önskemålen på marginalerna relativt högt, så att de sedan kan sänkas om systemets beteende visar sig tillfredsställande vid flygtesterna.

Mätdata från flygtester visar att det tar cirka 9-10 sekunder att genomföra en hastighetsförändring från 12 till 20 m/s vid full motoreffekt. Den önskade stigtiden för hastigheten skulle därför kunna tänkas vara lite över tio sekunder. Nedan följer de önskade marginalerna samt utfallet från optimeringen.

	Önskemål	Utfall
Fasmarginal	45°	46,8°
Amplitudmarginal	8,0 dB	8,0 dB
Stigtid	10 sekunder	14,9 sekunder
Maximal översläng	10 %	4,6 %

Tabell 4.8: Önskemål och resultat från optimeringen av hastighetsregulatorn.

PID	Kp	Ki	Kd	N	Max	Min
Ensam PID	1,20	0,15	-13,1	0,065	100	0

Tabell 4.9: Parametrar från optimeringen av hastighetsregulatorn.

Som kan avläsas i tabell 4.8 så blev den optimerade stigtiden något längre än den önskade, men i övrigt uppfylldes kraven.

5

Systemimplementering

Det här kapitlet beskriver hur mjukvaran i drönaren och det kringliggande stödsystemet har implementerats. Först förklaras mjukvaruarkitekturen, hur drönarens program är uppdelat i *lager* och *moduler*. Sedan beskrivs hur dessa lager är strukturerade i en hierarki för att uppnå ett system som ämnar vara enkelt att anpassa till liknande hårdvara. Därefter följer ett avsnitt som förklarar systemets telemetridel, och beskriver hur och varför gruppen valt att överföra data via en databas, samt vilka egenskaper databasen har och översiktligt hur den fungerar. För att förenkla kommunikation med drönaren upprättades också ett användargränssnitt i form av en Android-applikation. Dess funktioner och utformning behandlas senare i kapitlet innan slutligen gruppens utförande av praktisk testning och datainsamling beskrivs.

5.1 Mjukvaruarkitektur

I det här avsnittet behandlas hur mjukvaran i drönaren är uppbyggd i en lagerstruktur för att eftersträva målet om generalitet. Först beskrivs övergripande hur de olika lagren samspelar och därefter varje lagers ansvarsområde.

Mjukvaran är skriven i programmeringsspråket *Java*, som valdes av flera anledningar. Java tillåter att arbete med projektet kan utföras från flera olika operativsystem. Utöver detta är språket starkt typat vilket innebär minskad risk för mjukvarufel då typfel fångas vid kompilering istället för vid körning. Det kan även använda externa bibliotek för att kommunicera med hårdvara. Det centrala kriteriet för val av programmeringsspråk var att det ska främja målet om generalitet. Eftersom Java tillät gruppen att strukturera koden i ”inkapslade” moduler, där varje modul fungerar som en separat entitet, underlättades en eventuell anpassning av programmet till andra flygsystem. Även testning underlättas när varje modul fungerar självständigt, vilket är att föredra vid ett iterativt upplägg av arbetet.

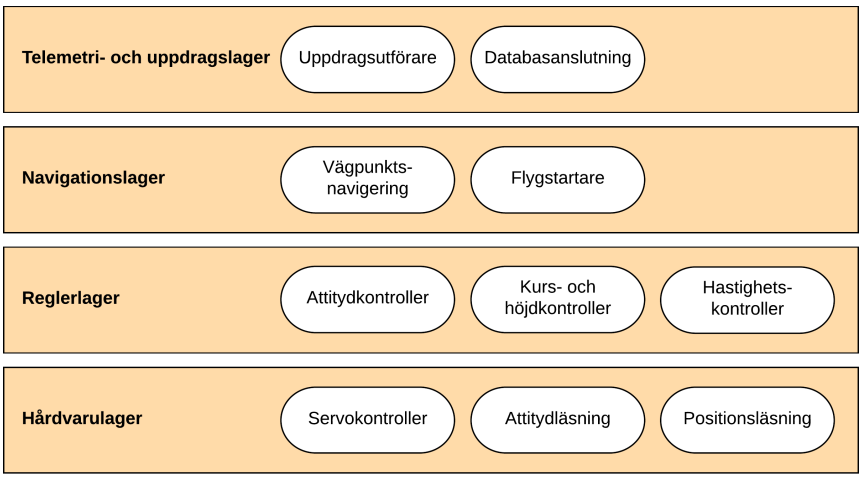
Modulerna i mjukvaran strukturerades enligt en lagerseparerad struktur, vilket illustreras i figur 5.1.

En lagerseparerad mjukvaruarkitektur är en förlängning av den modulbaserade strukturen. En fördel med den lagerspereradade mjukvaruarkitekturen är bland annat att den är lätt att förstå och implementera [45]. En annan fördel är att lagren är oberoende av varandra, vilket innebär att man enkelt kan ändra saker vid ett lager utan att behöva oroa sig över de andra [46].

Kodbasen blev således starkt hierarkisk, eftersom lagren kommunicerar endast nedåt. Moduler nära hårdvaran sägs ha *låg nivå*, medan moduler med mer abstrakt övergripande ansvar sägs vara på *hög nivå*.

Styrsystemet designades nerifrån och upp, där hårdvaruspecifika moduler utvecklades först, vilka sedan kunde användas av moduler högre upp i arkitekturen, till exempel styrningsmoduler i reglerlagret. Att moduler på låg nivå utvecklades först gjorde att systemet kunde testas och verifieras del för del och användas direkt, enligt ett arbetssätt som kallas *horisontell implementering* [47]. När moduler på lägre nivå kopplades ihop med moduler högre upp i systemet kunde de högre modulerna lättare testas, då de lägre modulerna redan var verifierade. Detta gjorde också att ingen logik som redan var implementerad i en lägre modul behövde återskapas högre upp i strukturen, då alla lager bygger på lagren under.

I kommande avsnitt följer mer detaljerade beskrivningar av varje lagers syfte och uppgifter, vilket även illustreras med exempel på modulernas samverkan under användning. För en visualisering som beskriver de övergripande modulernas samverkan och kommunikationsriktning, se bilaga C.



Figur 5.1: Diagram över den lagerseparerade mjukvaruarkitekturen.

5.1.1 Hårdvarulager

I botten av lagerstrukturen ligger hårdvarulagret, där alla moduler som kommunicerar direkt med, eller har stark koppling till, hårdvara är samlade. Dessa moduler har direkt åtkomst till olika kommunikationsmedier på huvuddatorn och tillhandahåller ett *gränssnitt* för moduler högre upp i systemet att styra och läsa från hårdvara. Eftersom hårdvarulagret kapslar in all hårdvaruspecifik kommunikation abstraheras detaljer som inte är relevanta bort i högnivå-kod, vilket minskar de högre lagrens beroende av hårdvarans egenskaper och konfiguration.

Av de två gränssnitten som tillhandahålls av hårdvarulagret är ett avsett för sensorer och ett för styrning av servomotorer samt huvudmotorn. Sensorgränssnittet erbjuder metoder som hämtar det aktuella värdet från en sensor samt tidsstämpeln för mätvärdet, och metoder som tillåter andra klasser i koden att prenumerera på nya sensorvärden. Det vill säga att detta gränssnitt följer *observer-pattern* i Java. Sensorgränssnittet använder sig av *Java generics*, som tillåter sensorvärdets typ att definieras för varje ny implementering, vilket var en positiv egenskap då alla sensorer kunde abstraheras till samma gränssnitt.

Gränssnittet för servo- och motorstyrning erbjuder metoder som tar in antalet grader som

ett roder skall vinklas och hur många procent av den maximala effekten som skall ges för motorn.

Denna inkapsling av moduler gjorde att logiken för funktionerna inte behövde återskapas högre upp i lagerstrukturen. Detta gjorde det lättare att kalibrera varje servo individuellt, vilket är väsentligt för att flygplanet ska kunna styras korrekt. Kalibreringen gjordes manuellt genom att ange olika styrsignaler och mäta motsvarande rodervinkel-utslag.

5.1.2 Reglerlager

För att kunna styra flygplanet smidigt med hänsyn till dess tillstånd implementerades en struktur med regulatorer. Reglermoduler i detta lager är starkt baserade på designen av reglersystemet i avsnitt 4.3. I programmet för styrningen av drönaren implementerades reglersystemen som ett eget lager, *reglerlagret*, som är högre upp i lagerhierarkin än hårdvarulagret. Reglerlagret kommunicerar nedåt i lagerstrukturen och begär vinklar för olika roder och effektprocent till motorn, utan att känna till hur dessa sedan sätts. Detta ger reglerlagret en viss frikoppling från hårdvaran och gör det kompatibelt med liknande uppsättningar av komponenter.

Att reglerlagret verkar oberoende av hårdvarukonfigurationen är ännu en del i strävan efter generalitet, vilket som tidigare nämnts grundar sig i projektets syfte och mål. Utöver att lättare kunna överföras till andra system tillåter frikopplingen att hårdvara kan bytas ut och kalibreras om i det befintliga systemet, vilket innebär att systemet kan kalibreras om utan att ändra på mjukvara.

Reglermodulernas beteende anpassas genom inmatning av reglerparametrar via externa filer. Detta gör det lättare att konfigurera beteendet hos reglersystemet, eftersom inga ändringar behöver göras direkt i koden.

Varje modul i reglerlagret implementerar dessutom ett gränssnitt som erbjuder metoder för att styra reglermodulen utifrån drönarens nuvarande tillstånd. När en regleringsmodul läser flygplanets tillstånd kan den beräkna de rodervinklar som krävs för att orientera planet i korrekt riktning. Ett exempel på detta är attityd-modulen i regleringslagret som försöker bibehålla en begärd attityd utifrån drönarens tillstånd.

Att reglersystemen använder sig av samma gränssnitt gör att de även kan kopplas ihop för att uppnå mer avancerade kontrollsystem. Detta används till exempel vid implementering av höjdreglering, och tillåter regulatorstrukturen som visas i figur 4.4 i avsnitt 4.3.1 att lättare programmeras in i mjukvaran. Detta är ännu ett sätt på vilket gruppen arbetat med horisontell implementering, eftersom en regulator kan utvecklas och verifieras för att sedan användas i utvecklingen av ett större regulatorsystem.

5.1.3 Navigationslager

Navigationslagret ansvarar för navigationsbeteende hos flygplanet, och ligger en nivå högre än reglerlagret. Navigationslagret använder regleringslagret för att kontrollera flygplanets riktning och attityd i navigationen. Navigationslagret kommunicerar inte direkt med hårdvarulagret och därför återanvänds logiken som redan har implementerats längre ner i lagerstrukturen. Navigationslagret behöver därför endast innehålla logik för att kunna navigera efter vägpunkter, starta och landa.

Gränssnittet som finns i reglerlagret implementeras även av modulerna från navigations-

lagret. Detta medför att moduler i navigationslagret kan utökas och kombineras på samma sätt som de i reglerlagret.

5.1.4 Telemetri- och uppdragslager

Det högsta lagret i arkitekturen innehåller all logik för telemetri och uppdragshantering. Detta lager ansvarar för att bestämma hur uppdrag ska utföras, samt för att meddela databasen om drönarens nuvarande status.

Moduler i detta lager försöker analysera drönarens tillstånd, och utifrån detta ändra navigationsmetod om det behövs för att slutföra uppdraget eller för att undvika en oönskad situation. Det som skiljer detta lager från undre lager är att lägre lager styr planet utan att analysera flygplanets tillstånd. Om planet hamnar i ett oönskat tillstånd, som snabb nedgång, kommer dessa moduler fortsätta försöka utföra sina uppgifter, även om det riskerar att drönaren havererar. I detta lager ska moduler känna till omständigheterna som planet befinner sig i och byta navigationsläge när det är lämpligt.

En av dessa moduler, uppdragsmodulen, har övergripande ansvar för att genomföra ett helt uppdrag från start till landning. Den kontrollerar drönarens navigationsläge, bestämmer hur det aktuella uppdraget ska utföras och anropar relevanta navigationsmoduler för att uppfylla sitt mål. När uppdragsmodulen aktiveras använder den en modul som nollställer all styrning på flygplanet för att inte orsaka skada. Efter att en operatör har gett startkommando och kastat drönaren anropas startmodulen som startar motorn. Sedan driver drönaren rakt fram med maximal tillåten stigvinkel tills den når en fördefinierad höjd. När startsekvensen är genomförd kallar uppdragsmodulen på nästa navigationsmodul, som börjar utföra uppdraget och navigerar genom alla vägpunkter. När uppdraget är avklarat byter uppdragsmodulen den aktiva navigationsmodulen till en modul som navigerar flygplanet tillbaka till sin startpunkt. Om ett nytt uppdrag ges kommer uppdragsmodulen börja genomföra detta istället. Efter att drönaren har tagit sig tillbaka till startpunkten går uppdragsmodulen in i den sista fasen. Här kommer uppdragsmodulen använda sig av en landningsmodul och landa på en förbestämd plats.

Lagerstrukturen möjliggör för de olika nivåerna att samspela. Moduler i telemetri- och uppdragslagret uppskattar vilket tillstånd drönaren befinner sig i och ger utifrån detta navigationslagret i uppgift att styra drönaren på lämpligt sätt. Dessa tillstånd kan exempelvis vara start, landning, eller vägpunktsnavigering. Moduler i navigationslagret ger moduler i reglerlagret i uppgift att orientera flygplanet i en viss riktning för att följa uppdraget. Reglerlagret avgör hur stora vinklar som behöver anges ut till varje roderservo för att uppnå rätt orientering. Roderutslag skickas vidare till moduler i hårdvarulagret som översätter det till en styrsignal för hårdvaran. Denna lagerstruktur och dess beroende mellan modulerna illustreras med ett komponentdiagram i bilaga C.

5.2 Telemetri och databasintegrering

För att drönaren skulle kunna skicka och ta emot telemetri, innehållande till exempel uppdrag, flygzoner och statusuppdateringar, behövdes ett system för kommunikation. Huvuddatorn som användes har ett inbyggt WiFi genom vilket en trådlös anslutning till ett nätverk kan skapas. Om nätverket delades från en extern enhet, till exempel en dator eller mobil, kunde huvuddatorn ansluta till detta. För att sedan kunna överföra telemetri behövdes en koppling mellan huvuddatorn och enheten skapas. Då det från enheten

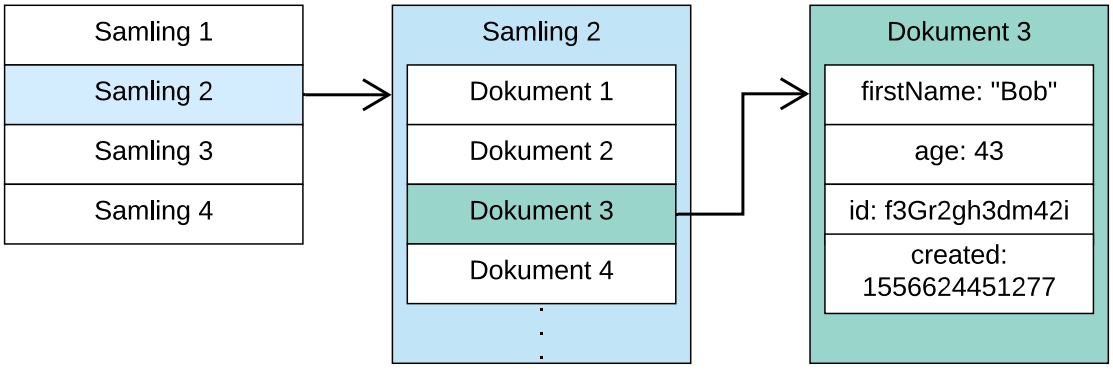
gick att se huvuddatorns IP-adress kunde en anslutning etableras via SSH (Secure Shell, ett protokoll för säkra anslutningar över internet), och kommunikation med drönaren blev möjlig. Detta fungerade så länge drönaren och enheten var i närheten av varandra, men då projektet syftar till att fungera över långa distanser behövdes en mer tillförlitlig lösning.

Eftersom 3G-täckningen är bra på de flesta platser i Sverige [48] valdes detta slutligen som det använda mediet, varpå en 3G-dongel kopplades till huvuddatorn. Genom denna fick drönaren en pålitlig internetanslutning. Det problem som tidigt upptäcktes med denna lösning var att det inte längre gick att ansluta med SSH. Då enheten inte längre var i samma nät som drönaren fungerade inte direktanslutningar, även om dess IP-adress var känd, eftersom de publika 3G-näten använder så kallad NAT-teknik (Network Address Translation) [49]. Detta gör att drönarens riktiga IP-adress är ”gömd” inom det publika nätverket, och den adress som syns är delad med många andra enheter, vilket ledde till att en annan lösning behövde hittas.

För att kringgå detta användes en ”mellanhand” som både enheten och drönaren kunde ansluta till, och däremellan skicka data till varandra. Lösningen som valdes var *Cloud Firestore*, en NoSQL-databas från Google Firebase, gjord för lagring av data.

5.2.1 Databasens uppbyggnad

I en NoSQL-databas lagras informationen annorlunda än i traditionella relationsdatabaser i tabellform [50]. I Firestore används så kallade dokument och samlingar. Dokumenten är de som innehåller datan, och ett eller flera dokument organiseras under en samling. Detta illustreras i figur 5.2. Med den här typen av databas kan de olika typerna av telemetri och data lagras i olika samlingar. Exempelvis sparades uppdrag och statusuppdateringar från drönaren i separata samlingar, vilket gjorde databasen mer lättorganiserad än om allt skulle varit i samma samling. Strukturen för all data och telemetri som lagrades i databasen kan ses i bilaga B.



Figur 5.2: Databasstrukturen i Firestore.

5.2.2 Kommunikation och lagring av data från drönaren

I och med att telemetrin går via en databas går det inte att kommunicera ”direkt” mellan drönaren och en enhet. Kommunikationen fick ske genom att den enhet som skickade data, till exempel ett uppdrag, först skickade det till databasen som lagrade det varpå drönaren läste från databasen och på så sätt fick det nya uppdraget.

Alla databaser i Firestore är publika som standard och tillgängliga via en URL. För att inga obehöriga ska få åtkomst till databasen och dess innehåll är det möjligt att tillämpa regler som definierar vem som har tillgång till vad. Dessa regler sattes så att endast inloggade användare får läsa och skriva i databasen. Problemet med detta var att det då krävdes autentisering för drönaren och andra enheter.

Google Firebase tillhandahåller även andra funktioner och tjänster så som *Authentication*, *Cloud Functions* och *Cloud Messaging*. Authentication är en tjänst för att autentisera användare, till exempel för åtkomst till Firestore. Cloud Functions används som molnserver för Node.js-funktioner. En Node.js-funktion i Cloud Functions är en JavaScript-funktion som kan ta emot och returnera data, ungefär som en hemsida där en användare fyller i formulär som sedan skickas till webbservern. När en funktion skapas får den en unik URL från vilken den kan anropas med. Cloud Messaging används för att skicka notifikationer till mobila enheter.

Dessa fyra tjänster; Firestore, Authentication, Cloud Functions och Cloud Messaging, finns som klientbibliotek att använda i till exempel Java, Android och iOS. Med Authentication- och Firestorebiblioteket skulle det gå att direkt läsa och skriva till databasen. Men då dessa bibliotek testades på drönaren visade det sig att programvaran på Raspberry Pi inte hade stöd för HTTPS, vilket biblioteken krävde för att skapa säkra anslutningar till Google Firebase. För att lösa detta ersattes biblioteken med Cloud Functions-funktioner, vilka endast kräver en HTTP-anslutning [51]. De Node.js-funktioner som skapades listas och förklaras i bilaga D.

När drönaren skickar eller läser data från databasen anropas en av dessa funktioner. Denna lösning gjorde att latensen (tiden från det att funktionen anropas tills det att funktionen utförts och skickat tillbaka ett svar) ökade med någon tiondels sekund, men den förflyttade samtidigt en del beräkningar till Cloud Functions vilket medförde mer utrymme för drönaren att utföra andra operationer.

Då kommunikation mellan drönaren och databasen fungerade behövdes även ett enkelt sätt för en användare att kunna se drönarens statusuppdateringar, till exempel dess position på en karta, samt modifiera databasen. För att göra detta lättare skapade gruppen således ett användargränssnitt.

5.2.3 Användargränssnitt

Det beslutades att användargränssnittet skulle vara i form av en Android-app, där det skulle vara möjligt att interagera med drönaren. För att förenkla utvecklingen av appen skapades en lista med punkter på vad som skulle ingå. Dessa var:

1. En karta för att kunna se drönarens position och följa vart den har flugit
2. En lista med statusuppdateringar från drönaren för att se dess värden, exempelvis hastighet och höjd över havet
3. Kunna skapa en flygsession där det går att se alla statusuppdateringar inom en viss tidsperiod
4. Kunna skapa uppdrag i form av en lista med geografiska koordinater och sätta det som aktivt för drönaren att utföra
5. Kunna ändra drönarens variabler och värden, till exempel säga åt den att starta eller uppdatera sitt uppdrag

6. Kunna skapa zoner och ändra landningszonen för drönaren
7. Ta emot notifikationer, exempelvis när drönaren når en punkt i ett uppdrag eller har landat

För design och utveckling av appen användes Android Studio, en utecklingsmiljö för Android-applikationer utgiven av Google. Till skillnad mot drönaren har Android stöd för HTTPS vilket medförde att biblioteken för Authentication, Cloud Messaging och Firestore kunde integreras i appen. För att autentisera användaren i appen användes så kallade *anonyma konton*. Med detta menas att inga inloggningsuppgifter behövdes, utan användaren fick ett anonymt konto skapat automatiskt när appen startade, vilket gav tillgång till databasen. Samtidigt som användaren loggades in skickades enhetens ID-token, skapad av Cloud Messaging, till databasen. På så sätt kunde notifikationer tas emot från drönaren.

De sju tidigare nämnda punkterna delades upp och resulterade i nio olika vyer i appen, vilka förklaras i bilaga E.

5.3 Upplägg av flygtester för att verifiera resultat av systemimplementation

För att verifiera att den implementerade mjukvaran fungerade utfördes flygtester med prototypen. De två centrala mål som testades under flygtesterna var att lyfta och att navigera mellan vägpunkter. Dessa mål testades först individuellt genom att byta mellan lägen via radiosändare, och när båda hade verifierats testades de tillsammans i sekvens.

Test av autonom start utfördes normalt av två personer. En person satte drönaren i startläge, vilket innebar att klaffen förlängdes och mjukvaran började reglera för att lyfta. Den andra personen höll i flygplanet och när motorn varvat upp så kastade personen flygplanet framåt, varpå flygplanet lyfte. På den mobila applikationen övervakades drönarens höjd och när höjden nådde 30 meter avslutades testet. Drönaren växlades då till manuellt läge och landades av operatören.

För att testa drönarens förmåga att autonomt navigera mellan vägpunkter angavs först de önskade vägpunkterna i mobilapplikationen. För att undvika problem med flygplanets begränsade svängradie placerades vägpunkterna runt 200 meter ifrån varandra och höjden angavs till 40 meter. Uppdraget sattes sedan som aktivt i mobilapplikationen och drönaren startades manuellt från marken. Operatören styrde upp till ett lämpligt utgångsläge på ungefär 30 meters höjd, där automatiken startades och navigation utfördes autonomt. Efter avslutat uppdrag tog operatören över kontrollen och landade planet manuellt.

Under testflygningarna samlades en stor mängd data från både sensorer och styrsystem. Data från de sista flygningarna presenteras senare i resultatavsnittet.

6

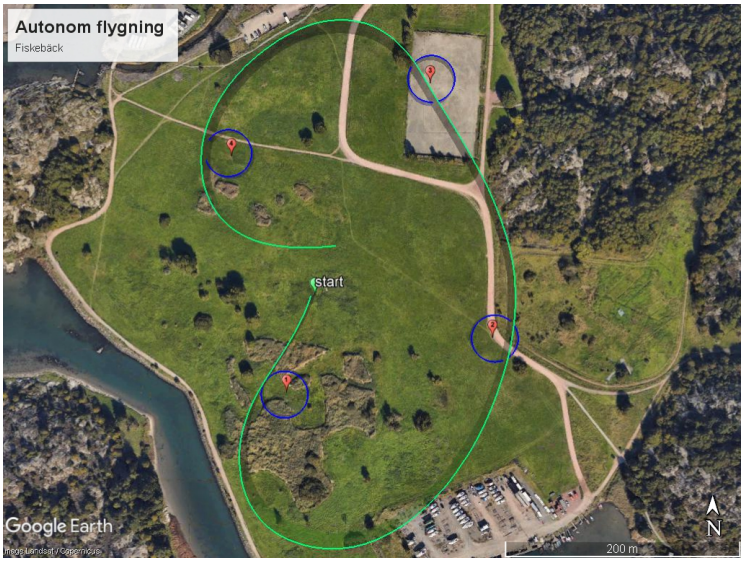
Resultat

I detta kapitel redovisas resultat från utförda flygtester, samt hur väl projektet uppfyllde de satta målen. Kapitlet är indelat i olika avsnitt som var för sig beskriver hur väl flygplanet presterat ur olika aspekter.

6.1 Flygvägsföljning

Ett av huvudmålen med projektet var att kunna navigera autonomt mellan fördefinierade vägpunkter. En vägpunkt ansågs passerad när flygplanet passerat inom en 20 meters radie från vägpunkten. Flygplanets prestanda med hänseende till flygvägsnavigation har utvärderats vid flera olika flygtester.

I figur 6.1 visas flygplanets färdväg under ett flygtest, tillsammans med uppdragets vägpunkter och deras tillhörande målcirklar för att indikera toleransradie.

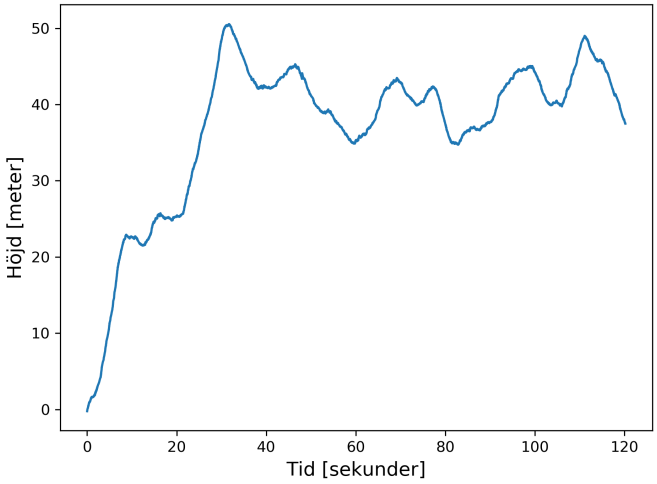


Figur 6.1: Autonom flygning utförd vid Sjöbacken Fiskebäck. Den långa linjen visar drönarens flygning. Markeringarna med cirklar indikerar utsatta vägpunkter och deras toleransradie visas med cirkeln. Flygbild © 2018 Google.

Som visas i figur 6.1 är flygplanet nära på att missa flera av vägpunkterna. Vid något flygtest missades den första vägpunkten, vilket ledde till att flygplanet fick vända och åka tillbaka för att nå vägpunkten och sedan kunna fortsätta uppdraget.

6.2 Höjdhållning

Flygplanets förmåga att hålla höjden undersöktes vid flertalet testflygningar.



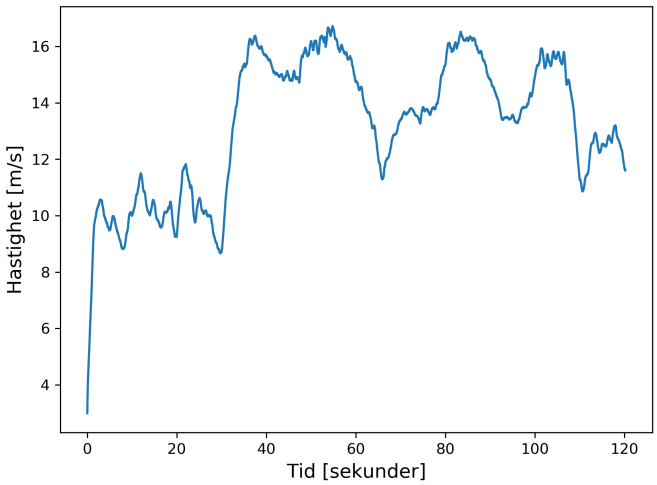
Figur 6.2: Diagram som visar flygplanets höjd under flygning med autonom start och vägpunktsföljning.

I figur 6.2 syns flygplanets loggade höjddata under en flygning. Under den här flygturen sköttes både start och navigation autonomt av styrsystemet. Ojämnheten i kurvan vid omkring 17 sekunder beror på att systemet här byter från att hålla en konstant pitch och motoreffekt, som den gör i startsekvensen, till att istället reglera efter målhöjd och hastighet. Det är även här flygplanet börjar svänga mot sin första vägpunkt. Målhöjden under just det här uppdraget var satt till 40 meter, och när målhöjden nåddes fortsatte flygplanet att stiga till cirka 50 meter innan det återigen lade sig kring cirka 40 meters höjd. Under den autonoma navigationen, mellan ca 40-120 sekunder håller sig dock drönaren inom ungefär åtta meter från målhöjden. I avsnitt 7.1.3 kan man läsa om de slutsatser som dras utifrån dessa resultat.

6.3 Hastighetshållning

Under samma test som höjdhållningen utvärderades testades även flygplanets prestanda gällande hastighetshållning.

Som visas i figur 6.3 aktiverades hastighetsregleringen vid omkring 30 sekunder och reglersystemet arbetade för att hålla en konstant hastighet på 15 m/s genom luften. Notera att hastigheten vid flera tillfällen, till exempel kring 70 sekunder, avvek från börvärdet på 15 m/s. Den lägsta hastighet som uppmättes under navigation mellan vägpunkterna var cirka 11,5 m/s, vilket är ungefär 23% lägre än börvärdet. Slutsatser utifrån dessa resultat presenteras senare i avsnitt 7.1.4.



Figur 6.3: Diagram som visar flygplanets hastighet under autonom start och flygning.

6.4 Start och landning

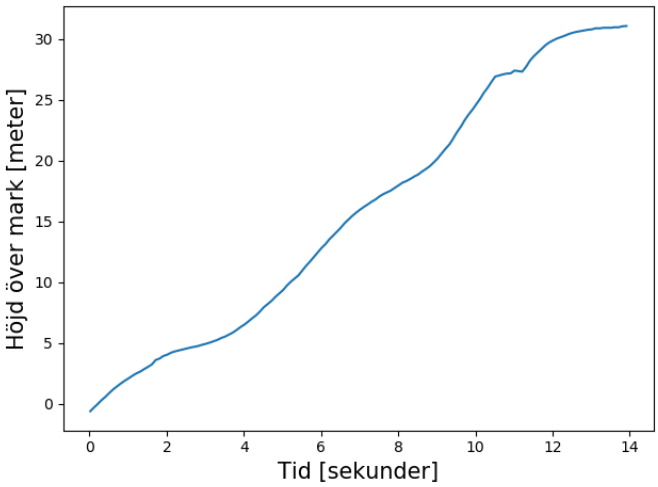
Som kan ses i avsnitt 1.4 var det relativt låg prioritet på målen om autonomt reglerad start och landning. Landning togs upp för diskussion när styrsystemet började bli färdigt, men togs inte vidare på grund av komplexiteten i implementationen, och alltså finns inga resultat att redovisa gällande detta. När ett flyguppdrag är avklarat ges en notis till operatören via applikationen, och flygplanet söker sin hempunkt tills den antingen får ett nytt uppdrag eller tills operatören tar över kontrollen och landar den manuellt.

Vad gäller start fanns ett önskemål om att drönaren skulle kunna reglera sig själv under stigningen, men gruppen förbehöll sig möjligheten att en operatör skulle behöva kasta eller på annat sätt tillföra kraft till flygplanet för att få upp det i luften. Detta har gruppen lyckats med genom att drönaren, då operatören har gett startkommando via användargränssnittet och kastat den, reglerar sig själv för att bibehålla vingarna vågräta med maximal, av reglersystemet tillåten, stigvinkel och hög statisk motoreffekt tills den når en hårdkodad höjd (30 meter), varpå den går över i uppdragsläge. Figur 6.4 visar hur drönarens höjd ökar någorlunda linjärt med två meter i sekunden under start.

Under testflygningar kunde man visuellt observera hur flygplanet ibland vid autonom start drev av från den initiala kursen på grund av vind från sidan.

6.5 Mjukvarans generalitet och anpassningsbarhet

Den nuvarande mjukvaruarkitekturen tillåter framtida utvecklare att bygga på de existerande abstraktionsnivåerna som finns i de olika lagren. Sensorer kan bytas ut utan att mycket mjukvara behöver ändras och nya sensorer kan integreras in i mjukvara med hjälp av de existerande gränssnitten. Om utvecklare vill implementera nya navigations- eller regleringsmetoder kan de återanvända moduler från existerande lager. Detta gör att utvecklare slipper läsa in sig på lågnivåkommunikation och reglering om de ska implementera funktioner högre upp i arkitekturen.



Figur 6.4: Förändring av drönarens höjd under start. Vid elva sekunder byttes läge till vägpunktsnavigation, vilket är anledningen till ojämnheten i kurvan.

6.6 Kommunikation och användargränssnitt

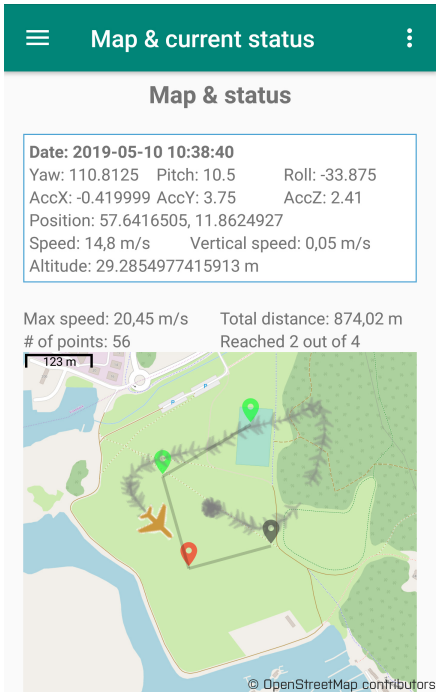
Kommunikationen mellan applikationen och drönaren via databasen (vars implementation redovisas i 5.2) har visat sig fungera tillförlitligt utan avbrott. Drönaren skickar statusuppdateringar till databasen en gång i sekunden, vilka sedan överförs till den mobila applikationen. Detta ger operatörer möjligheten att följa drönarens position på en karta i realtid, vilken visas i figur 6.5. Dessutom sparas alla tidigare flygningar som har gjorts i databasen, vilka sedan kan nås genom den mobila applikationen. Samtliga av de listade punkterna i avsnitt 5.2.3 implementerades i applikationen. Fler skärmdumpar av andra vyer i applikationen kan ses i bilaga E.1.

6.7 Undvikande av geografiskt definierat område

Undvikande av geografiskt definierat område var ett av projektets mål vilket prioriterades lågt till förmån för funktioner som ansågs viktigare. Arbetet på undvikande av fördefinierade zoner började därför relativt sent i projektet och blev inte färdigställt. Appen utrustades med funktionalitet för att mata in en zon, definierad av hörnpunkter inmatade i systemet av operatör, vilken kunde skickas med i ett uppdrag tillsammans med vägpunkter. En algoritm för själva undvikanden av zoner påbörjades, men färdigställdes aldrig.

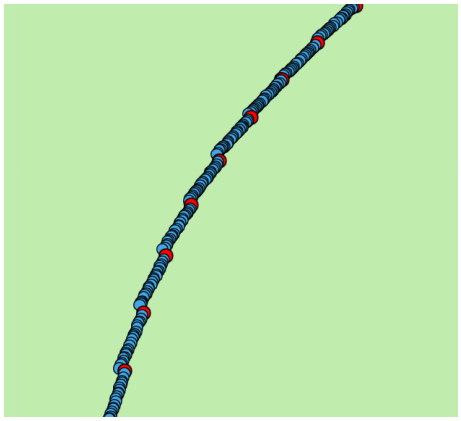
6.8 Resultat av Kalmanfiltrerad positionsinläsning

I figur 6.6 visas en flygrutt genomförd med positionsinläsning från GPS, där mätdata från den Kalmanfiltrerade positionsinläsningen samlats in. Som kan utläsas från figuren följer positionsinläsningen från Kalmanfiltret den från GPS:en väl, men några värden avviker vilket diskuteras i avsnitt 7.2.2. Kalmanfiltret ger ungefär 20 gånger fler positionsvärden per sekund än GPS-sensorn, vilket är svårt att se i figur 6.6, men illustreras i figur 6.7.



Figur 6.5: Skärmdump från appen då drönaren utför ett uppdrag.

Figur 6.6: Partiell flygväg ritad av flygtur med GPS-baserad navigation representerad i svart, med mätdata från Kalmanfiltrerad positionsinläsning markerad i vitt. Flygbilder ©2018 Google



Figur 6.7: Annan delsträcka från flygturen i figur 6.6. De blå punkterna representerar värden från Kalmanfiltret och de röda värden från GPS:en. Bild skapad med copypastemap.com. Karta © 2019 Google.

7

Diskussion och slutsatser

I denna sektion återfinns reflektioner över vald metodik och hur det påverkade resultatet och målsättningarna för projektet. Först diskuteras resultat från projektet och vissa slutsatser dras kring olika upplägg som gruppen valt, varpå förslag ges på hur projektet skulle kunna utvecklas i framtiden.

7.1 Diskussion kring resultat i förhållande till målsättningarna

I följande avsnitt diskuteras de resultat som relaterar till målsättningarna som ställs upp i avsnitt 1.4.

7.1.1 Kommunikation till och från drönaren

Hur kommunikation med drönaren genomförs i slutprodukten redovisas i avsnitt 6.6.

Att använda en databas som mellanhand mellan operatör och drönare visade sig vara ett bra val, eftersom detta tillät flera användare att ta emot datan i realtid. Detta kan nyttjas vid drönaruppdrag med behov av samverkan mellan flera användare, som till exempel leveranser. Det underlättade också testning av systemet under utvecklingsstadiet. Den här implementationen av kommunikation medför också att operatörer får möjligheten att byta drönarens uppdrag under flygning, vilket ofta inte är möjligt vid direktanslutning mellan användargränssnitt och drönarenhet.

En möjlig nackdel med denna implementation är att alla personer som har mobilapplikationen kan skicka kommandon och uppdrag till drönaren. En vidareutveckling för att förhindra detta kan vara att tilldela användare olika behörigheter, till exempel operatör och observatör.

Applikationen är i nuläget också endast utrustad för att styra en drönare åt gången. Ett annat sätt att utveckla det här projektet blir således att möjliggöra styrning av flera drönare åt gången, och monitorera deras flygningar simultant.

7.1.2 Navigation mellan vägpunkter

I avsnitt 6.1 redovisas att drönaren kunde navigera mellan vägpunkter. En observation som kan göras är hur nära drönaren är att missa vägpunkterna 1, 2 och 4, även om de har en toleransradie på 20 meter. Denna osäkerhet vid navigation tros bero på flera faktorer. En av dessa tros vara att reglersystemet reglerar långsamt, något som programmerats in

för att ge ett stabilt system utan för ryckiga svängar. Drönarens rollvinkel begränsas också till att maximalt kunna ge 20°, även detta för att inte skapa ryckigt beteende, men att höja denna gräns skulle kunna göra det lättare att nå vägpunkterna med större säkerhet. Man skulle också kunna sänka hastigheten, vilket skulle göra det lättare att passera vägpunkter som är nära varandra eftersom svängradien minskar vid långsammare flygning.

En annan faktor som skulle kunna påverka drönarens förmåga att navigera till vägpunkter är algoritmen som används för att beräkna begärd kurs. Den nuvarande implementationen för kursnavigation, som är beskriven i avsnitt 4.3.2, navigerar till en vägpunkt genom att begära kurs rakt mot den. Detta innebär att inflygningsbanan inte blir rak om drönaren utsätts till exempel för sidvindar, i och med att drönaren driver ur kurs men samtidigt fortsätter rikta sig mot målet. Desto närmare drönaren är vägpunkten, desto snabbare kommer kursfel växa om drönaren påverkas på detta vis. Eftersom drönaren är begränsad till sin svängradie, kan det hända att drönaren missar vägpunkten om kursfelet ändras för snabbt.

En kurvad inflygning är inte heller den kortaste vägen mellan två punkter, vilket kan vara önskvärt av olika anledningar. En alternativ metod för att navigera mellan vägpunkter skulle vara att implementera en linjeföljningsalgoritm, som håller drönaren på linjen som går mellan två vägpunkter och på så sätt minskar risken att drönaren flyger förbi vägpunkten på grund av drift i sidled samtidigt som kortaste inflygningsväg garanteras.

7.1.3 Höjdhållning

Som en del av navigationen reglerar också systemet sin höjd, resultat av vilket redovisas i avsnitt 6.2. Som kan avläsas i figur 6.2 avvek höjden under den autonoma navigationen med uppemot åtta meter från det av operatör begärda värdet på 40 meter. Detta tros bero på att drönaren svänger kring dessa tidpunkter, vilket innebär att kraften på flygkroppen minskar i vertikalled då drönaren lutar. Reglersystemet har alltså inte reagerat tillräckligt snabbt för att kunna förhindra att planet avviker från sin målhöjd under svängar.

Man kan även observera att flygplanet, under den autonoma starten, vid omkring 30 sekunder skjuter igenom målhöjden på 40 meter, fortsätter upp till nästan 50 meter innan den etablerar sig kring målhöjden.

Utifrån dessa två nämnda avvikelser konstateras att även om flygplanet lyckades genomföra sitt uppdrag så finns det fortfarande förbättringspotential gällande höjdregleringen. Det hade därför varit intressant att undersöka alternativa parametrar och regulatorstrukturer för att se om dessa hade presterat bättre.

7.1.4 Hastighetsreglering

I avsnitt 6.3 presenterades resultat gällande systemets förmåga att reglera hastigheten. Avvikelser från målhastigheten på 15 m/s kunde observeras, vilket kan ses i figur 6.3. Här kan ett samband dras mellan avvikelserna i hastighet och förändringar i höjd, vilket kan ses om man jämför figur 6.3 med figur 6.2. När till exempel höjden ökade med ungefär tio meter vid 70 sekunder kunde man samtidigt observera att hastigheten sjönk med cirka fyra m/s. Man kan utifrån detta dra slutsatsen att höjdförändringar har en relativt stor påverkan på flygplanets hastighet, vilket kan vara rimligt eftersom en ökad anfallsvinkel leder till större luftmotstånd och således bromsar flygplanet.

Även om systemet är stabilt och håller hastigheten relativt väl, så hade det varit önskvärt

med en snabbare regulator som ökat motoreffekten redan innan hastigheten hunnit minska väsentligt, för att få ett system med stabilare hastighet.

Man kan även tänka sig att den framtagna hastighetsmodellen, som endast beskriver sambandet mellan motoreffekt och hastighet, hade kunnat utökas till att innefatta även sambandet mellan pitch och hastighet. Utifrån detta hade man sedan kunnat designa en ny regulator som till exempel proportionellt höjer motoreffekten när pitchvinkeln ökar, och på så sätt förebygga hastighetsminskningen.

7.1.5 Autonom start och landning

Som nämns i avsnitt 6.4 lyckades gruppen inte att implementera autonom landning. Anledningen till detta var bland annat att ny hårdvara hade behövts som på ett mer exakt sätt kunde bestämt drönarens höjd över marken. Detta skulle således kunna vara ett sätt att vidareutveckla projektet för att åstadkomma en mer holistisk navigationssekvens.

Autonom start implementerades däremot och visade goda resultat vid testflygningar. Som nämns i avsnitt 6.4 kunde det observeras att flygplanet ibland drev av från sin initiala kurs. I implementationen av startsekvensen beordras flygplanet att hålla vingarna horisontella. För att undvika att flygplanet driver av från sin tänkta startkurs, skulle man istället kunna ändra så att flygplanet bibehåller en specificerad kurs, eller färdlinje.

7.1.6 Generalitet

Vilken grad av generalitet och anpassningsbarhet som mjukvaran lever upp till är svårt att mäta. Vid två tillfällen under projektets gång gavs dock tecken på att mjukvaran till viss del är generell. Första gången var när en höjdsensor slutade fungera och lätt kunde bytas ut. Modulen i hårdvarulagret som läste från den gamla sensorn behövde ändras, men övriga moduler och lager förblev desamma som innan. En annan del av projektet som framlyfter mjukvarans generalitet var vid framtagningen av regulatorparametrar. Parametrarna kan anges externt via filer istället för att vara hårdkodade, vilket gjorde det enkelt att testa dem vid flygning. Detta låter operatörer kalibrera reglersystemet för liknande flygfarkoster utan att ändra i kod, och när de aerodynamiska egenskaperna hos flygplanet i det här projektet ändrades kunde även dessa enkelt kalibreras för.

Därav kan man till viss del motivera att mjukvaran uppfyller det uppställda syftet som bland annat säger att ”Mjukvaran ska kunna ligga till grund för framtida utveckling och forskning som ämnar nyttja autonoma drönare”, vilket målet om generalitet är satt för att tillfredsställa.

7.1.7 Undvika fördefinierat geografiskt område

I resultatavsnitt 6.7 nämndes att undvikande av zoner inte hann bli implementerat fullt ut. Detta har inte medfört några hinder vid flygtesterna då dessa utförts på väl valda områden och under ständig uppsikt. För att drönaren i framtiden ska kunna utföra flygningar helt autonomt över större områden behöver dock denna funktionalitet färdigställas och verifieras.

7.2 Kalmanfilter

Testerna visade att den Kalmanfiltrerade positionsinläsningen hade gett mer tillförlitliga värden om accelerometern hade fungerat bättre. De visade dock att Kalmanfiltret fungerade bra, men det fanns inte tid nog inom ramarna för projektet för att anpassa och implementera det i programvaran. Som nämns i avsnitt 3.2 upptäcktes i projektets slutskede ett sätt att läsa in GPS-data ofta nog för att styrsystemet skulle fungera ändå.

7.2.1 Kalmanfiltrets kovariansmatriser

Anledningen till att accelerometern visade en bättre standardavvikelse än GPS:en i avsnitt 3.3.2 är troligen att drönaren låg inomhus under datainsamlingen och därmed fick dålig signal eftersom GPS fungerar sämre inomhus (vilket nämns i 2.3). Det är även möjligt att accelerometern ger en mindre standardavvikelse vid stillastående än vid rörelse.

Ett annat problem som kan ligga bakom de undermåliga resultaten för testet kan vara att endast ett prediktionssteg utfördes per observationssteg. Eftersom osäkerheten för estimeringen byggs upp för varje prediktion skulle accelerometern få mindre trovärdighet vilket skulle kunna leda till en bättre estimering.

7.2.2 Utfall av Kalmanfiltrerad positionsinläsning

De avstickande värdena som syns i figur 6.6 i avsnitt 6.8 antas bero på fel hos accelerometern eftersom filtret i figur 3.3 samt i simuleringen av Kalmanfiltret i avsnitt 3.3.3 gav indikation på att filtret fungerade bra.

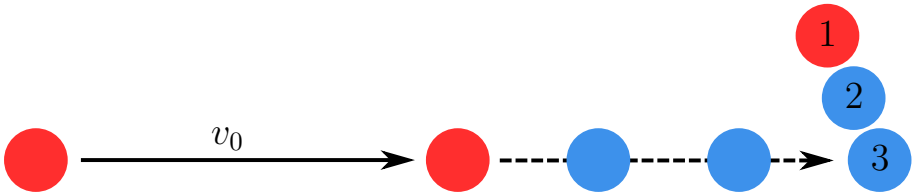
Som beskrevs i avsnitt 3.3.2 hade även accelerometern problem med att medelvärdet inte var noll vilket ytterligare minskade dess trovärdighet.

Enligt figur 6.7 verkar accelerometerns värden inte bidra särskilt mycket vid svängar, vilket tycks underligt i och med att flygplanet påverkas av centripetalkraft i detta läge. Beteendet liknar snarare det som demonstreras i figur 7.1. Detta kan dels bero på effekten som visas i figur 3.4c, det vill säga att eftersom accelerometern i dessa fall har en försumbar inverkan på hastigheten kommer den uppskattas till samma hastighet som tidmätningen mellan de två senaste GPS-koordinaterna gav indikation på.

Man skulle därmed möjligen kunna designa ett Kalmanfilter som ignorerar accelerometern helt, eftersom den ändå tycks ha såpass låg inverkan. Detta skulle innebära att prediktionssteget endast består av antagandet att flygplanet kommer fortsätta i hastigheten uppmätt mellan de två tidigare GPS-koordinaterna. Därmed kommer filtret, vid observation, ge en estimerad position baserat på observationen samt den tidigare hastigheten. Se figur 7.1 för visualisering. I detta fall skulle möjligen osäkerheten för estimeringen bli något högre eftersom hastigheten och riktningen ändras vid svängar.

För framtida vidareutveckling av Kalmanfiltret med accelerometer skulle en bättre metod för att estimera $\mathbf{Q}$ och $\mathbf{R}$ behöva användas. Just nu är de satta enligt ekvation 3.5. Eftersom filtret verkade fungera bra med dem satta till ekvation 3.4, dvs mindre förtroende för GPS-sensorn, är det rimligt att tro att accelerometerns inverkan på positionen vid korrektionssteget är helt försumbar. Dock kommer accelerometern fortfarande ha inverkan på prediktionssteget. Se figur 3.4c.

Utöver detta skulle det första testet i sektion 3.3.2 kunna göras om med flera predik-



Figur 7.1: Visualisering av positionsestimering med kalmanfiltret utan accelerometern. Här representerar de blåa prickarna kalmanfiltrets estimeringar och de röda dess observationer. Filtret kommer, under prediktionsfasen, utgå ifrån att drönaren fortsätter med samma hastighet (v_0). Vid observation kommer den observerade positionen (1) tillsammans med den senaste estimerade positionen (3) och deras osäkerheter skapa en ny estimerad position(2).

tionssteg för varje korrektionssteg enligt figur 2.5. Detta eftersom flera estimeringar ökar osäkerheten hos accelerometern, vilket är önskvärt eftersom filtret då inte behöver ta lika mycket hänsyn till dess värde.

7.2.3 Analys av behovet av Kalmanfilter

Anledningen till att Kalmanfiltret designades var som nämnt i avsnitt 3.3 att gruppen ville åtgärda den låga uppdateringsfrekvensen från den ursprungliga GPS-konfigurationen, samt få bättre mätsäkerhet på positionsvärdena.

Precisionen hos GPS-sensorn visade sig vara nog hög för att genomföra målen med projektet, och när sedermera inställningen för att ta emot GPS-data med en frekvens av 10 Hz upptäcktes blev det designade Kalmanfiltret något obsolet, då det visade sig vara nog för att styrsystemet skulle fungera tillfredsställande. Således var det ingen förlust för projektet att detta inte integrerades i slutprodukten, eftersom Kalmanfiltrerad positionsinläsning presterar på mycket högre nivå än vad som krävs av det system som utvecklats här.

7.3 Slutsatser

Utgångspunkten för projektet har varit att skapa och implementera mjukvara för autonom styrning och långdistanskommunikation för bevingade luftfarkoster. Projektets delmål har till flera punkter uppfyllts. Den slutgiltiga prototypen klarar bland annat av att autonomt starta, stiga till målhöjd och sedan navigera mellan förutbestämda vägpunkter. Under flygningen kan dessutom operatören övervaka drönarens status i realtid med hjälp av en utvecklad mobilapplikation. Med applikationen kan man även under pågående flygning uppdatera drönarens uppdrag.

Projektet har inte lyckats med att implementera mjukvara för autonom landning. Landningarna har vid flygtesterna istället utförts helt manuellt. Utveckling av mjukvara för att undvika zoner har påbörjats, men är inte färdigställd eller testad. Anledningen till att delmålen om autonom landning och zonundvikande inte har uppnåtts beror främst på att de övriga delmålen har tagit mer tid än förväntat, vilket gjort att delmålen med lägst prioritet fick skjutas framåt.

Som förslag på framtida utveckling lämnas framför allt implementation av autonom land-

ning och undvikande av hinder i olika utsträckning, dels undvikande av fördefinierade geografiska zoner, exempelvis områden där flygförbud gäller, men även undvikande av objekt i flygvägen, exempelvis höga hus eller berg. Även autonom start från startbana är en tänkbar utveckling, så att autonomi genom hela flygprocessen, från start till landning, uppnås.

Det hade också varit intressant att utforska om det skulle kunna vara möjligt att tillämpa samverkan mellan flera drönare i en grupp. Till exempel samverkan vid avsökning av ett område, eller samarbete vid logistik.

Litteratur

[1] D. Segerson. Explosionsartad ökning av drönare, [Online], URL: https://sverigesradio.se/sida/artikel.aspx?programid=91&artikel=6259792 (hämtad 2019-02-02).

[2] Drönarcentralen. Vad är en drönare?, [Online], URL: https://xn--drnarcentralen-wpb.se/kunskapsbanken/vad-ar-en-dronare/ (hämtad 2019-02-02).

[3] Intelligent Logistik. Amazon tar patent på ”fågelholkar” för drönare, [Online], URL: http://intelligentlogistik.com/nyhetsflode/itteknik/amazon-tar-patent-pa-fagelholkar-for-dronare/ (hämtad 2019-02-02).

[4] M. Borak. E-commerce giant JD will build tens of thousands delivery drone landing pods, [Online], URL: https://technode.com/2018/02/05/e-commerce-giant-jd-will-build-tens-of-thousands-delivery-drone-landing-pods/ (hämtad 2019-04-02).

[5] Miljönytta, ”Drönare hittar nya transportvägar”, 2015, [Online]. URL: https://miljonytta.se/framtid/dronare-hittar-nya-transportvagar/ (hämtad 2019-02-02).

[6] Droneify, ”Inspektion med drönare”, 2019, [Online]. URL: https://www.droneify.se/inspektion/ (hämtad 2019-02-02).

[7] CompassDrone. Phantom 4 Pro Datasheet. Centennial, CO, USA: DJI, [Online], URL: https://compassdrone.com/phantom-4-pro-datasheet/ (hämtad 2019-04-02).

[8] J. Warell. ”Drönare” i *Nationalencyklopedin*, [Online], URL: http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/dr%C3%B6nare (hämtad 2019-03-11).

[9] H. H. Perrit, Jr. och E. O. Sprague, *Domesticating Drones: The Technology, Law and Economics of Unmanned Aircraft*. London, Great Britain: Routledge, 2016.

[10] TSFS 2017:110. Transportstyrelsens föreskrifter om obemannade luftfartyg, [Online], URL: https://www.transportstyrelsen.se/TSFS/TSFS%202017_110.pdf (hämtad 2019-02-10).

[11] A. Vacca och H. Onishi, ”Drones: military weapons, surveillance or mapping tools for environmental monitoring? The need for legal framework is required”, *Transportation Research Procedia*, årg. 25, s. 51–62, 2017. DOI: 10.1016/j.trpro.2017.05.209. URL: https://doi.org/10.1016/j.trpro.2017.05.209.

[12] Skyline Software Systems, inc. Yaw, Pitch and Roll Angles, [Online], URL: http://www.skylinesoft.com/SkylineGlobe/TerraExplorer/v6.5.0/APIReferenceGuide/Yaw_Pitch_and_Roll_Angles.htm (hämtad 2019-05-16).

[13] R. Axelsson, R. Danewid och I. Larsson, *Segelflyg: en lärobok*. Skövde: Segelflygsport, 1996.

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

[15] National Oceanic and Atmospheric Administration. Magnetic Declination, [Online], URL: https://www.ngdc.noaa.gov/geomag/declination.shtml (hämtad 2019-05-16).

[16] Nationalencyklopedin. GPS, [Online], URL: https://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/gps (hämtad 2019-05-02).
[17] Lantmäteriet. Frågor och svar om GPS, [Online], URL: https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Fragor-och-svar/?faq=b160 (hämtad 2019-04-02).
[18] H. Fredrick. Why Doesn't GPS Work Inside a Building?, [Online], URL: https://itstillworks.com/doesnt-gps-work-inside-building-18659.html (hämtad 2019-05-10).
[19] Djexplo. File:Latitude and Longitude of the Earth.svg, [Online], URL: https://commons.wikimedia.org/wiki/File:Latitude_and_Longitude_of_the_Earth.svg (hämtad 2019-05-16).
[20] A. Becker. Kalman filter overview, [Online], URL: https://www.kalmanfilter.net/default.aspx (hämtad 2019-05-14).
[21] T. Harada, H. Uchino, T. Mori och T. Sato, "Portable orientation estimation device based on accelerometers, magnetometers and gyroscope sensors for sensor network", i *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI2003.*, IEEE. DOI: 10.1109/mfi-2003.2003.1232655. URL: https://doi.org/10.1109/mfi-2003.2003.1232655.
[22] S. Srini. The Kalman Filter: An algorithm for making sense of fused sensor insight, [Online], URL: https://towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e (hämtad 2019-05-01).
[23] G. Bishop, G. Welch m. fl., "An introduction to the kalman filter", *Proc of SIGGRAPH, Course*, årg. 8, nr 27599-23175, s. 41, 2001.
[24] S. Lobdell. GPS and Accelerometer Sensor Fusion with a Kalman Filter, a Practical Walkthrough (Golang), [Online], URL: http://scottlobdell.me/2017/01/gps-accelerometer-sensor-fusion-kalman-filter-practical-walkthrough/ (hämtad 2019-05-02).
[25] M. Nilsson. Kalman Filtering with Unknown Noise Covariances, [Online], URL: https://www.diva-portal.org/smash/get/diva2:1041201/FULLTEXT01.pdf (hämtad 2019-05-02).
[26] M. Saha, B. Goswami och R. Ghosh. Two novel metrics for determining the tuning parameters of the Kalman Filter, [Online], URL: https://arxiv.org/pdf/1110.3895.pdf (hämtad 2019-05-02).
[27] G. Czerniak. Kalman Filters for Undergrads Part I: Linear Kalman Filters, [Online], URL: http://greg.czerniak.info/guides/kalman1/ (hämtad 2019-05-02).
[28] L. Korb, *Memories of the Apollo and Space Shuttle Programs*. Page Publishing, Inc., 2017.
[29] C. Q. Choi. Strange but True: Earth Is Not Round, [Online], URL: https://www.scientificamerican.com/article/earth-is-not-round/ (hämtad 2019-06-07).
[30] D. R. Williams. Earth Fact Sheet, [Online], URL: https://nssdc.gsfc.nasa.gov/planetary/factsheet/earthfact.html (hämtad 2019-06-07).
[31] B. Lennartsson, *Reglerteknikens grunder*. Studentlitteratur, 2011, vol. 4.
[32] A. Saberi, Z. Lin och A. Teel, "Control of linear systems with saturating actuators", *IEEE Transactions on Automatic Control*, årg. 41, nr 3, s. 368–378, mars 1996. DOI: 10.1109/9.486638. URL: https://doi.org/10.1109/9.486638.
[33] Android. Geomagnetic Field, [Online], URL: https://developer.android.com/reference/android/hardware/GeomagneticField.html (hämtad 2019-05-02).

[34] RS Calibration. All You Need to Know About Sensor Calibration, [Online], URL: https://www.rscal.com/all-you-need-to-know-about-sensor-calibration/ (hämtad 2019-06-07).

[35] D. Sachs. (2015). Sensor Fusion on Android Devices: A Revolution in Motion Processing, [Online], Google, URL: https://www.youtube.com/watch?v=C7JQ7Rpwn2k&t=23m20s.

[36] J. B. Høstmark, "Modelling Simulation and Control of Fixed-wing UAV: CyberSwan", examensarb., Norwegian University of Science och Technology, juli 2007. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/259855/348487_FULLTEXT01.pdf (hämtad 2019-04-28).

[37] S. S. Hamada, "Development of a Small Unmanned Aerial Vehicle Longitudinal Model for Future Flutter Testing", examensarb., Embry-Riddle Aeronautical University, maj 2018. URL: https://commons.erau.edu/edt/395/ (hämtad 2019-04-28).

[38] L. Ljung, *System identification : theory for the user.* Prentice Hall, 1999, ISBN: 0-13-656695-2.

[39] T. Shafer, S. Viken, N. M. Favaregh, C. H. Zeune, N. Williams och J. Dansie, "Comparison of Computational Approaches for Rapid Aerodynamic Assessment of Small UAVs", i *52nd Aerospace Sciences Meeting*, American Institute of Aeronautics och Astronautics, jan. 2014. DOI: 10.2514/6.2014-0039. URL: https://doi.org/10.2514/6.2014-0039.

[40] *Principles of Flight*, 4. utg. Nordian AS, 2009, ISBN: 9788281070900.

[41] The Airfoil Investigation Database. Rhode St. Genese 32, [Online], URL: http://www.airfoildb.com/airfoils/193 (hämtad 2019-03-13).

[42] The Airfoil Investigation Database. S9033 (7.5%), [Online], URL: http://www.airfoildb.com/airfoils/134 (hämtad 2019-03-13).

[43] HobbyKing. Turnigy™ TG9d Digital Micro Servo 21T 1.8kg / 0.09sec / 9g, [Online], URL: https://hobbyking.com/en_us/turnigytm-tg9d-digital-micro-servo-1-8kg-0-09sec-9g.html (hämtad 2019-05-01).

[44] R. W. Beard och T. W. McLain, *Small Unmanned Aircraft : Theory and Practice.* Princeton University Press, 2012, ISBN: 9781400840601.

[45] R. Gupta. Benefits and Drawback of a Layered Architecture, [Online], URL: https://www.pixelstech.net/article/1493900728-Benefits-and-Drawback-of-a-Layered-Architecture (hämtad 2019-05-16).

[46] M. Richards. Layered Architecture, [Online], URL: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html (hämtad 2019-05-16).

[47] J. Virgo. Understanding Software Development with Vertical Slices vs Horizontal Slices, [Online], URL: https://aptude.com/blog/entry/understanding-software-development-with-vertical-slices-vs-horizontal-slices/ (hämtad 2019-05-16).

[48] Telia. Täckningskartor, [Online], URL: https://www.telia.se/privat/support/tackningskartor (hämtad 2019-04-29).

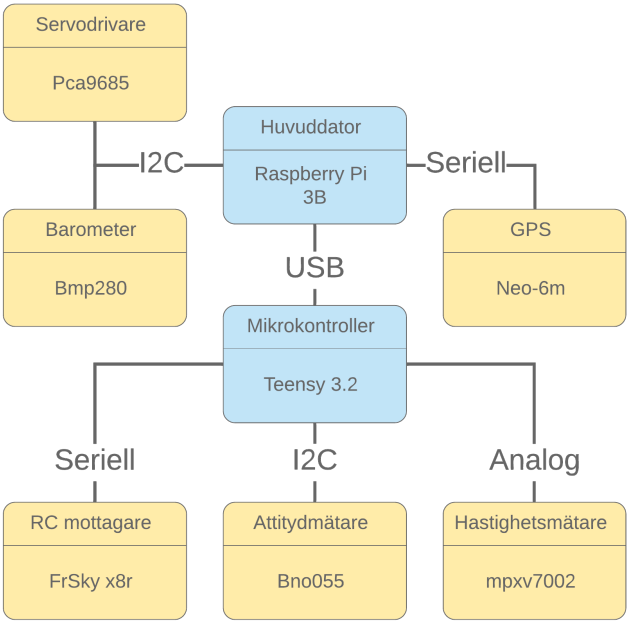
[49] D. Wing, "Network Address Translation: Extending the Internet Address Space", *IEEE Internet Computing*, årg. 14, nr 4, s. 66–70, juli 2010. DOI: 10.1109/mic.2010.96. URL: https://doi.org/10.1109/mic.2010.96.

[50] C. Strauch, *NoSQL databases*, Föreläsningsanteckningar, 2011. URL: https://christof-strauch.de/nosqldbs.pdf.

[51] Google. Call functions via HTTP requests, [Online], URL: https://firebase.google.com/docs/functions/http-events (hämtad 2019-05-01).

A

Delkomponenter



Figur A.1: Kopplingschema mellan sensorer och datorer

B

Databasens struktur

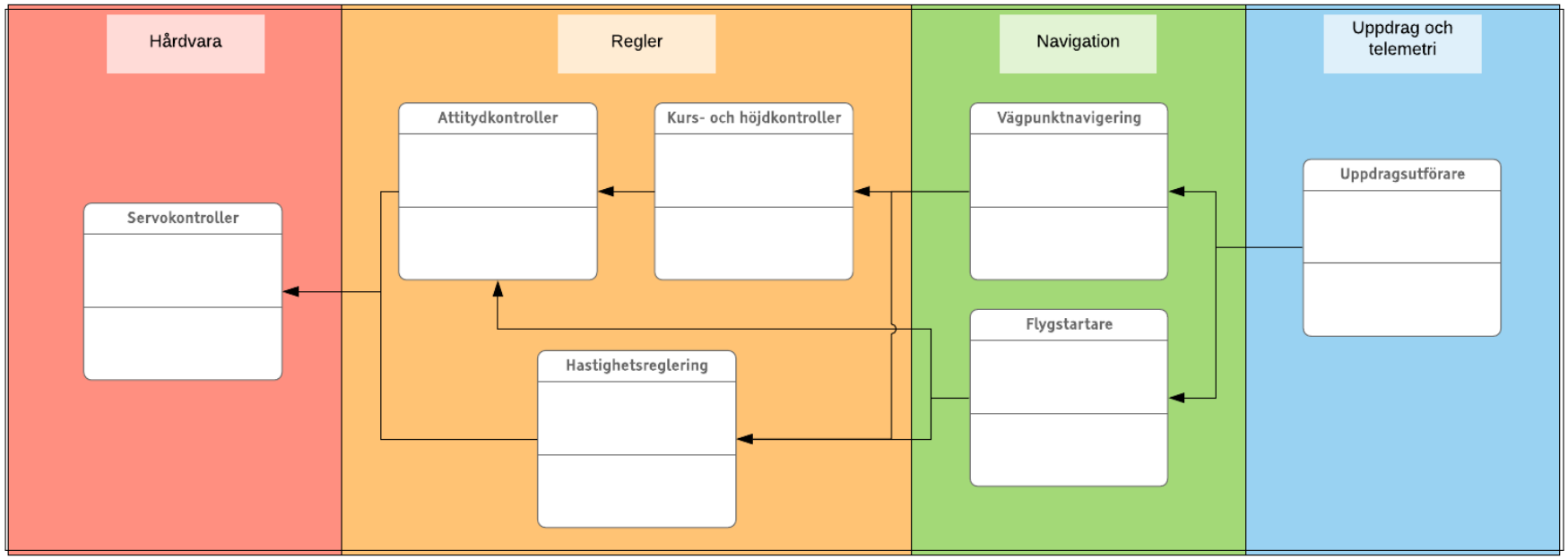
Samlingar är i fetstil och dokument i kursiv där dess data visas i en underlista. Dokumentnamn i versal som börjar med ett *$*-tecken får unika identifikationssträngar i databasen när de skapas. Dessa kan det finnas flera av i en samling, till exempel en för varje uppdrag.

- **missions** → *$MISSION_ID*: skapade uppdrag
  - `name`: namn på uppdrag
  - `points`: lista med vägpunkter som ska besökas
    - `latitude`: en vägpunkts latitudkoordinat
    - `longitude`: en vägpunkts longitudkoordinat
  - `altitude`: lista med vägpunkternas altitud, en för varje i `points`
    - `alt`: höjd över havet i meter
- **zones** → *$ZONE_ID*: skapade zoner
  - `name`: namn på zonen
  - `active`: om denna zon är aktiv och ska användas av drönaren
  - `maxAlt`: maximal flyghöjd inom zonen
  - `minAlt`: minsta flyghöjd inom zonen
  - `type`: typ av zon. 0 = zonen har en minsta flyghöjd, 1 = zonen har en minsta och maximal flyghöjd, 2 = zonen har en maximal flyghöjd, 3 = flygförbud
  - `points`: lista med vägpunkter som utgör zonen
    - `latitude`: en vägpunkts latitudkoordinat
    - `longitude`: en vägpunkts longitudkoordinat
- **active** → *active*: uppdraget som drönaren ska utföra
  - `landingZone`: vald landningszon (*$ZONE_ID*)
  - `mission`: valt uppdrag (*$MISSION_ID*)
  - `reachedPoint`: antal vägpunkter som nåtts i uppdraget, 0 från början
- **flights** → *$FLIGHT_ID*: skapade flygningar
  - `name`: namn på flygning
  - `start`: tidpunkt för start av flygning
  - `end`: tidpunkt för slut av flygning
- **messagingID** → *$DEVICE_ID_TOKEN*: enhets-ID för app-notifikationer, se *Användargränssnitt*
  - `date`: när enhetsidentifikationen skapades
  - `token`: ID-strängen
- **status** → *$STATUS_ID*: statusuppdateringar från drönaren
  - `date`: när statusuppdateringar skapades
  - `alt`: drönarens höjd över havet
  - `lAccX`: linjär acceleration i X-led

  - `lAccY`: linjär acceleration i Y-led
  - `lAccZ`: linjär acceleration i Z-led
  - `roll`: rotation kring X-axeln
  - `pitch`: rotation kring Y-axeln
  - `yaw`: rotation kring Z-axeln
  - `pos`: drönarens GPS-position
    * `latitude`: punktens latitudkoordinat
    * `longitude`: punktens longitudkoordinat
- **values** → *values*: värden för att konfigurera drönaren
  - `goalTol`: en vägpunkts radie som drönaren måste vara inom för att nå punkten i ett uppdrag
  - `pressureRef`: tryckreferens vid start för att kalibrera höjdmätaren
  - `rudderDenominator`:
- **variables** → *variables*: variabler (true/false) för att konfigurera drönaren
  - `startFlight`: om drönaren ska starta och utföra uppdraget satt i **active** → *active*
  - `updateStatus`: om drönaren ska skicka statusuppdateringar till databasen
  - `missionUpdate`: om det finns ett nytt uppdrag för drönaren att hämta
  - `valuesUpdate`: om det finns nya värden för drönaren att hämta
  - `zoneUpdate`: om det finns nya zoner för drönaren att hämta

C

Mjukvarans lagerhierarki



Figur C.1: Diagram över moduler i olika lager och deras relation till varandra.

D

Node.js-funktioner

- updateStatus ← *status*[]
  Tar emot en lista med en eller flera statusuppdateringar. För varje status skapas ett dokument under **status**. När detta är klart returneras HTTP-statuskoden 201 (Created) och funktionen avslutas.
- getVariables → *variables*
  Läser först dokumentet från **variables** → *variables* och sparar det lokalt. Därefter sätts `missionUpdate`, `zoneUpdate` och `startFlight` till *false* i databasen. Detta görs för att förhindra att drönaren gör en operation, till exempel uppdatera uppdraget, flera gånger. Efteråt returneras dokumentet med variabler med en HTTP-statuskod av 200 (Done).
- getZones → *zone*[]
  Hämtar alla dokument från **zones** där `active` är *true*. Efter detta returneras en lista med dokumenten med en HTTP-statuskod av 200 (Done).
- getValues → *values*
  Hämtar dokumentet från **values** → *values* vilket returneras tillsammans med en HTTP-statuskod av 200 (Done).
- getMission → (*mission*, *zone*)
  Denna funktion hämtar det aktiva uppdraget och landningszonen. Först läses dokumentet från **active** → *active*. Därefter, om antingen `mission` eller `landingZone` inte existerar i dokumentet returneras en HTTP-statuskod 404 (Not found) utan någon data. Om båda existerar hämtas först uppdraget och sedan zonen från varsin samling (**missions** och **zones**), varpå dessa returneras med en HTTP-statuskod av 200 (Done).
- sendMessage ← (title, message)
  Tar emot en titel och ett meddelande vilka ska skickas som en notifiering till alla aktiva enheter med ett notifikations-ID i databasen. Om titeln eller meddelandet är tomt returneras 400 (Bad request). Annars hämtas de 100 senast skapade dokumenten från **messagingID**. Från varje dokument hämtas `token` och en notifikation skapas och sänds till enheten med denna ID-sträng med hjälp av Cloud Messaging. Sist, när alla notifikationer har skickats, sänds en HTTP-statuskod 200 (Done) tillbaks.

E

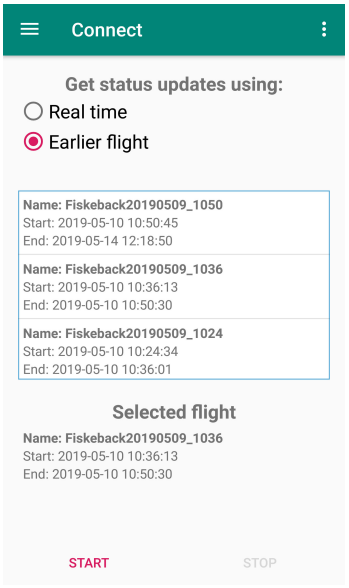
Android-appen

Här beskrivs de nio olika vyerna i appen:

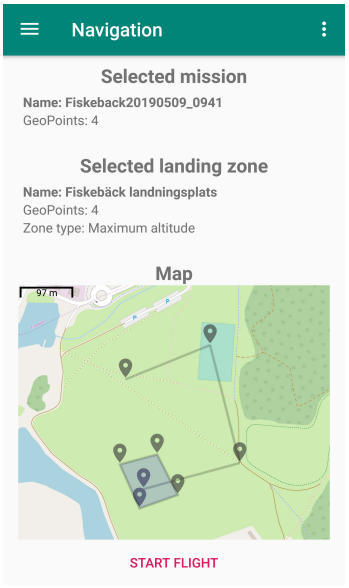
- *Connect* - Användaren kan välja mellan att få statusuppdateringar i realtid eller att välja en tidigare flygning. Om realtid väljes sätts `start` till den nuvarande tiden, annars till starttiden för flygningen. Därefter skapas en programloop som körs varje sekund vilken anropar databasen med en förfrågan om att hämta de 30 efterkommande statusuppdateringarna skapade efter `start`. Då varje dokument innehåller fältet `date` sätts `start` till det högsta (senaste) datumet, varpå 30 nya dokument hämtas nästa loop. På detta sätt itererar appen igenom det valda tidsintervallet och hämtar dokumenten i rätt ordning. Samtidigt som dokumenten hämtas räknar appen även ut den totala distansen som drönaren flugit och dess maxhastighet. Total distans fås genom att summera distansen mellan varje statusuppdaterings GPS-koordinat, och maxhastigheten genom att för varje statusuppdatering dividera distansen till den föregående GPS-koordinaten med tidsskillnaden mellan dem.
- *Map & current status* - I denna vy kan användaren följa drönarens position, aktiva zoner, landningszonen och uppdraget på en karta. Då loopen i *Connect* körs i bakgrunden uppdateras kartan en gång i sekunden med drönarens senaste position. Den senaste statusuppdatering som mottagits visas i en ruta ovanför kartan för att lätt kunna visa de övriga fälten i dokumentet, till exempel hastighet och höjd över havet. Här visas även maxhastighet och total distans.
- *Navigation* - Här kan användaren se namnen på det valda uppdraget som drönaren ska utföra och landningszonen. Dessa visas även visuellt på en karta.
- *Status list* - En lista vilken visar alla mottagna statusuppdateringar.
- *Variables* - En lista med alla variabler från databasen. Genom att trycka på en sak i listan ändras dess värde till det motsatta, exempelvis om `zoneUpdate` är *false* ändras den till *true*.
- *Values* - En likadan lista som i *Variables* men här med värden.
- *Flights* - Här kan användaren skapa, ändra och radera flygningar från en lista.
- *Zones* - En lista med alla zoner. Genom att trycka på en zon i listan kan användaren ändra om den ska vara aktiv eller inte, ange den som landningszon eller ändra den. Det går också att lägga till nya zoner genom att trycka på positioner på en karta.
- *Missions* - Här visas alla skapade uppdrag i en lista. Likadant som zonerna kan dessa raderas, ändras eller anges som aktivt uppdrag, och användaren kan även skapa nya uppdrag.

E.1 Skärmdumpar

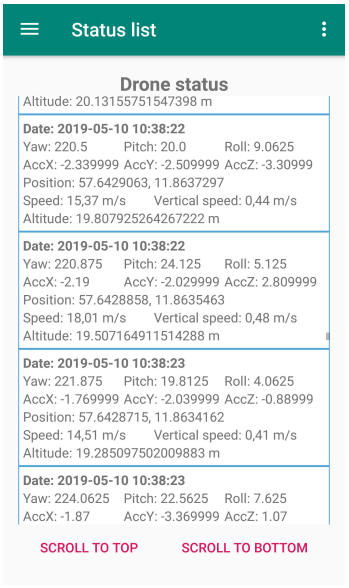
Figur E.1: Skärmdumpar från några av vyerna i appen.



(a) *Connect.*

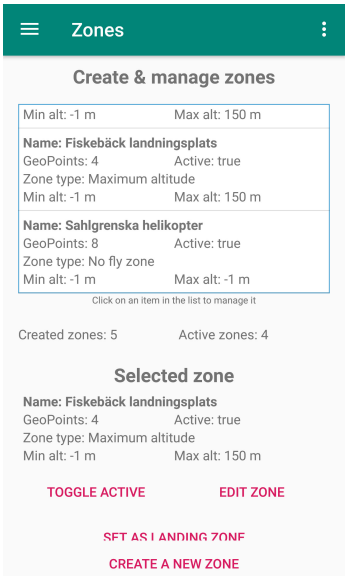


(b) *Navigation.*

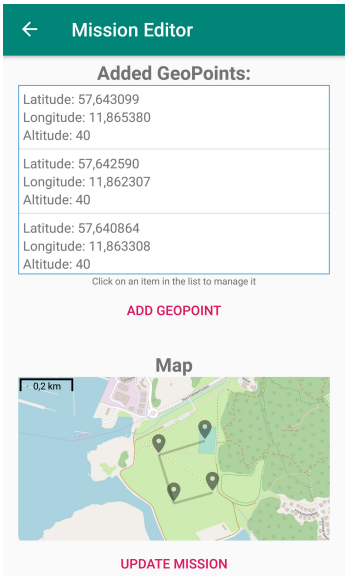


(c) *Status list.*

(d) *Variables.*



(e) *Zones.*



(f) *Mission editor.*

F

De framtagna systemmodellerna

F.1 Longitudinell modell

$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} -0.0157232 & 0.446417 & 0 & -9.81 \\ -1.27094 & -8.94544 & 14.228 & 0 \\ 0.000007 & -3.91826 & -4.00167 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0.019798 \\ 8.782982 \\ 53.38654 \\ 0 \end{bmatrix} \delta_e$$

u : Hastighet i längdaxelns riktning
w : Hastighet i lodaxelns riktning
q : Pitch rate, vinkelhastighet kring tväraxeln
θ : Pitch, vinkel kring tväraxeln
δ_e : Höjdrodervinkel

F.2 Lateral modell

$$\begin{bmatrix} \dot{v} \\ \dot{p} \\ \dot{r} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} -0.2772 & -0.0645413 & -15.2416 & 9.81 \\ -0.59353 & -16.1994 & 2.99126 & 0 \\ 0.736908 & -1.40881 & -0.435441 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v \\ p \\ r \\ \phi \end{bmatrix} + \begin{bmatrix} 2.13030 & -0.21894 \\ 0.845233 & -36.86 \\ -6.68573 & -5.76332 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_r \\ \delta_a \end{bmatrix}$$

v : Hastighet i tväraxelns riktning
p : Rotationshastighet kring längdaxeln
r : Rotationshastighet kring lodaxeln
ϕ : Vinkel kring längdaxeln
δ_r : Sidrodervinkel
δ_a : Skevrodervinkel

F.3 Motordynamik

Nedan presenteras den framtagna överföringsfunktionen mellan begärd motoreffekt och hastighet genom luften.

$$\frac{0,004505s + 0,001844}{s^6 + 0,8496s^5 + 1,003s^4 + 0,6829s^3 + 0,1766s^2 + 0,05313s + 0,00378}$$

G

Simulerad sensordata till Kalmanfiltret

Här presenteras data för simulation av Kalmanfiltret i avsnitt 3.3.3.

Notera att ny observation av positionen endast sker vid förändrat GPS-värde medan alla element bidrar till en estimering.

Simuleringsddata G.1: Datan för simulationen för figur 3.4a.


```
[
   {"timestamp":0,"gps_lat":57.687367983152775,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":0.2,"gps_lat":57.687367983152775,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":0.4,"gps_lat":57.687367983152775,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":0.6,"gps_lat":57.687367983152775,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":0.8,"gps_lat":57.687367983152775,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":1,"gps_lat":57.68737696630554,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":1.2,"gps_lat":57.68737696630554,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":1.4,"gps_lat":57.68737696630554,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":1.6,"gps_lat":57.68737696630554,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

   {"timestamp":1.8,"gps_lat":57.68737696630554,"gps_lon":11.978399,"abs_north_acc":0,"abs_east_acc":0},

```

```
22    {"timestamp":2,"gps_lat":57.68738594945831,"gps_lon":11.97
         8399,"abs_north_acc":0,"abs_east_acc":0}
23 ]
```

Simuleringsddata G.2: Datan för simulationen för figur 3.4b.

```
1  [
2     {"timestamp":0,"gps_lat":57.687359,"gps_lon":11.978399,"
         abs_north_acc":0,"abs_east_acc":0},
3
4    {"timestamp":0.2,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
5
6    {"timestamp":0.4,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
7
8    {"timestamp":0.6,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
9
10   {"timestamp":0.8,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
11
12   {"timestamp":1,"gps_lat":57.687367993070225,"gps_lon":11.97
       8399,"abs_north_acc":0,"abs_east_acc":0},
13
14   {"timestamp":1.2,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
15
16   {"timestamp":1.4,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
17
18   {"timestamp":1.6,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
19
20   {"timestamp":1.8,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
21
22   {"timestamp":2,"gps_lat":57.68737698614046,"gps_lon":11.978
       407411993931,"abs_north_acc":0,"abs_east_acc":1}
23 ]
```

Simuleringsddata G.3: Datan för simulationen för figur 3.4c.

```
1
2  [
3    {"timestamp":0,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
4
5    {"timestamp":0.2,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
```

```
6
7   {"timestamp":0.4,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
8
9   {"timestamp":0.6,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
10
11  {"timestamp":0.8,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
12
13  {"timestamp":1,"gps_lat":57.687367993070225,"gps_lon":11.97
       8399,"abs_north_acc":0,"abs_east_acc":0},
14
15  {"timestamp":1.2,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
16
17  {"timestamp":1.4,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
18
19  {"timestamp":1.6,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
20
21  {"timestamp":1.8,"gps_lat":57.687367993070225,"gps_lon":11.
       978399,"abs_north_acc":0,"abs_east_acc":1},
22
23  {"timestamp":2,"gps_lat":57.68737698614046,"gps_lon":11.978
       407411993931,"abs_north_acc":0,"abs_east_acc":0},
24
25  {"timestamp":2.2,"gps_lat":57.68737698614046,"gps_lon":11.9
       78407411993931,"abs_north_acc":0,"abs_east_acc":0},
26
27  {"timestamp":2.4,"gps_lat":57.68737698614046,"gps_lon":11.9
       78407411993931,"abs_north_acc":0,"abs_east_acc":0},
28
29  {"timestamp":2.6,"gps_lat":57.68737698614046,"gps_lon":11.9
       78407411993931,"abs_north_acc":0,"abs_east_acc":0},
30
31  {"timestamp":2.8,"gps_lat":57.68737698614046,"gps_lon":11.9
       78407411993931,"abs_north_acc":0,"abs_east_acc":0}
32 ]
```


Simuleringsddata G.4: Datan för simulationen för figur 3.4d samt 3.4e.


```
1  [
2   {"timestamp":0,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
3
4   {"timestamp":0.2,"gps_lat":57.687359,"gps_lon":11.978399,"
       abs_north_acc":0,"abs_east_acc":0},
5
```

```
  {"timestamp":0.4,"gps_lat":57.687359,"gps_lon":11.978399,"
     abs_north_acc":0,"abs_east_acc":0},

  {"timestamp":0.6,"gps_lat":57.687359,"gps_lon":11.978399,"
     abs_north_acc":0,"abs_east_acc":0},

  {"timestamp":0.8,"gps_lat":57.687359,"gps_lon":11.978399,"
     abs_north_acc":0,"abs_east_acc":0},

  {"timestamp":1,"gps_lat":57.687367993070225,"gps_lon":11.97
     8399,"abs_north_acc":0,"abs_east_acc":0},

  {"timestamp":1.2,"gps_lat":57.687367993070225,"gps_lon":11.
     978399,"abs_north_acc":0,"abs_east_acc":1},

  {"timestamp":1.4,"gps_lat":57.687367993070225,"gps_lon":11.
     978399,"abs_north_acc":0,"abs_east_acc":1},

  {"timestamp":1.6,"gps_lat":57.687367993070225,"gps_lon":11.
     978399,"abs_north_acc":0,"abs_east_acc":1},

  {"timestamp":1.8,"gps_lat":57.687367993070225,"gps_lon":11.
     978399,"abs_north_acc":0,"abs_east_acc":1},

  {"timestamp":2,"gps_lat":57.68737698614046,"gps_lon":11.978
     399,"abs_north_acc":0,"abs_east_acc":0},
]
```