CHALMERS

Plattform för autonom quadcopter

Kandidatarbete inom civilingenjörsprogrammen Elektroteknik och
Automation & Mekatronik

Trygve Grøndahl - Anton Johansson - Anton Olsson - Natalie Ternevi - Oscar Wellenstam

Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

Kandidatuppsats 2015:SSYX02-15-18

Plattform för autonom quadcopter

Trygve Grøndahl - Anton Johansson - Anton Olsson
Natalie Ternevi - Oscar Wellenstam

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Signaler och System
Chalmers Tekniska Högskola
Göteborg, Sverige 2015

Plattform för autonom quadcopter
Trygve Grøndahl - Anton Johansson - Anton Olsson
Natalie Ternevi - Oscar Wellenstam

Handledare: Patrik Bergagård, Instutitionen för Signaler och System
Examinator: Martin Fabian, Instutitionen för Signaler och System

Kandidatuppsats 2015:SSYX02-15-18
Institutionen för Signaler och System
Chalmers Tekniska Högskola
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Förord

Institutionen för Signaler och system vid Chalmers tekniska högskola tackas för tillhandahållet kandidatarbete.
Tack till handledare Patrik Bergagård, doktor vid institutionen för Signaler och system, för stöd och vägledning under projektet.
Vidare tackas Chalmers två intresseföreningar: Chalmers robotförening, CRF, och Elektrosektionens teletekniska avdelning, ETA. De har bistått med ett visat intresse och tillhandahållet materiel. Ett speciellt tack riktas till CRF:s gästfrihet och deras medlem Mikael Tulldahl som bistått med goda råd och donationer.

Trygve Grøndahl - Anton Johansson - Anton Olsson
Natalie Ternevi - Oscar Wellenstam
Göteborg, Maj 2015

Sammanfattning

En quadcopter är en flygfarkost utrustad med fyra kvadratiskt ordnade rotorer. Quadcoptrar kan användas för olika typer av uppdrag och öppnar upp för möjligheter att exempelvis utföra kartläggning av stora områden. Andra uppdrag där en quadcopter skulle kunna användas är då man önskar få en överblick över en speciell situation som vid brand eller vid stora folksamlingar. Med autonom styrning kan en quadcopter utföra avancerade uppgifter som inte hade varit möjliga att utföra genom manuell styrning.

Detta projekt har utförts på Chalmers tekniska högskola med syfte att utveckla en plattform för autonom styrning för en quadcopter. Under arbetets gång införskaffades en GPS med antenn, en ultraljudssensor, en magnetometer, en barometer, en multiplexer med tillhörande elektronikkomponenter och en Raspberry Pi.

Vid arbetets slut hade ovan nämnda komponenter implementerats i en integrerad plattform. Data från de olika sensorerna bearbetades i Rapsberry Pi:n som är plattformens beräkningsmässiga, centrala nod. Flera av komponenterna kommunicerar via program skrivna i programspråket C. Sensordata kan via programmen samlas in för att kunna användas för reglering i olika frihetsgrader och på så vis kunna utveckla autonom styrning.

Resultat ifrån de olika komponenternas individuella och gemensamma tester visar att de tillsammans erbjuder en plattform med stort potential. Höjdreglering implementerades vars resultat visar på en fungerande övergripande funktion mellan hård- och mjukvara. Med de uppnådda resultaten kan reglersystem för quadcopterns olika rotationsriktningar fortsättningsvis designas och implementeras för att erhålla önskad autonom funktion.

Resultat ifrån GPS:en visar att dess noggrannhet begränsar quadcopterns tänkta funktion och därmed användningsområde. Genom vidarutveckling och förbättringsarbete med olika komponenter samt utbyte av GPS-mottagare alternativt GPS-antenn kan troligtvis produktens funktion förbättras.

Nyckelord: plattform för autonom quadcopter, nätverks-rtk, Raspberry Pi, sensorhantering, GPS.

Abstract

A quadcopter, the vehicle in the project, is a flying aircraft equipped with four squarely ordered rotors. It was equipped with carefully selected components to create the best possible conditions to succeed with the project. The development of quadcopters opens new possibilities with a range of applications thanks to the vehicles stability and flexibility. Using autonomous control the quadcopter can complete assignments previously unfeasible.

This project has been carried out at Chalmers University of Technology with the aim to develop a platform for autonomous control of a quadcopter. To reach this goal a GPS with antenna, an ultrasound sensor, a magnetometer, a barometer, a multiplexer and a Raspberry Pi was purchased and implemented.

The finished product of the project is a platform, using a Raspberry Pi to compute the dataflow from multiple integrated sensors in programs written in the programming language C. The program can fetch sensor data from individual sensors at any time to be used in a desired application.

The collection of GPS data limits the applications on the quadcopter as the accuracy of the positioning is to low. This in turn limits the range of tasks able to be implemented on the platform. Through further development or exchanging the GPS module or its antenna, the opportunities in the platform can be improved.

The results from tests of the components both individually and collectively show that the product as a whole offers a platform for implementation in a quadcopter. Height regulation was implemented to show the overall combination of hard- and software. With the given results automatic control for the quadcopters other rotational directions can be implemented to achieve autonomy.

Keywords: platform for autonomous quadcopter, network-rtk, Raspberry Pi, sensor control, GPS.

Innehåll

1
Introduktion

För att få en grundläggande förståelse för rapportens innehåll kommer detta kapitel behandla projektets bakgrund och utsträckning. Dessutom kommer de viktigaste begreppen att förklaras. Inom området används ofta engelsk terminologi som här är översatt till svenska. Vidare ges en översiktlig bild av arbetsmetoden.

1.1 Bakgrund

En quadcopter är en typ av multirotor, även kallad multikopter, med fyra rotorer. De rotorer som sitter mitt emot varandra har samma rotationsriktning medan de resterande två roterar åt andra hållet. Detta medför att en quadcopters design eliminerar det moment som skapas av rotorerna vilket innebär att den mekaniska designen är enklare jämfört med traditionella helikopterutföranden med en ensam huvudrotor.

Ett stort användningsområde för quadcoptrar idag är fotografering från höga höjder. Detta används exempelvis av mäklare för visningsbilder och markägare för översiktibilder av skogsmarker. På sikt planeras användning av quadcoptrar inom den svenska polisen där de har till syfte att bland annat söka upp försvunna personer och övervaka demostationer. Polisens mål är att quadcoptrarna ska vara i drift under 2016 [56]. Vidare planerar företaget Amazon att i framtiden kunna erbjuda driektleverans av produkter till kunden med hjälp av quadcoptrar [57].

En autonom quadcopter är alltså en produkt med många potentiella användningsområden. Den autonoma styrningen är applicerbar i många sammanhang och kan möjliggöra både effektivisering och optimering av olika processer så som att kartlägga stora områden och att söka efter försvunna personer eller andra föremål. Med ökad tillgång till billig hårdvara och förbättrad prestanda ökar möjligheten att skapa avancerade system som kan utföra uppgifter som traditionellt gjorts manuellt eller inte gjorts alls.

En autonom quadcopters delprocesser kan exempelvis vara navigation, kollisionsundvikning, bildbehandling och identifiering av föremål vilka har stora användningsområden även utom ramen för en quadcopter. Processerna skulle kunna implementeras i flera olika sammanhang för att på så sätt skapa nya möjligheter för autonoma applikationer.

En möjlig tillämpning av en autonom quadcopter finns inom autonom sophämtning, för vilket Volvo AB driver ett projekt. Projektet bygger på möjligheten att effektivisera sophantering i nybyggda områden. En autonom quadcopter skulle kunna vara en del i ett större system där quadcoptern har som uppgift att identifiera och vidarebefordra soptunnors position till markrobotar som ska tömma dem.

1.2 Definitioner & förkortningar

I rapporten används ord och benämningar vars betydelse och innebörd behöver förklaras. Många begrepp och ord härstammar från engelskan men är här översatt till en svensk motsvarighet.

RF-sändare och RF-mottagare: Radiofrekvenskommunikation. Skickas och mottas med en 2.4 GHz signal.

Flygkontroller: Från engelska flight controller. En komponent vars uppgift är att upprätthålla quadcopterns grundläggande funktionalitet. Med grundläggande funktionalitet menas här att quadcoptern är manuellt manövrerbar, det vill säga att styrsignaler ifrån RF-mottagaren omvandlas till gaspådrag och att flygkontrollens interna sensorer och regulatorer åstadkommer önskad respons hos quadcoptern.

Motordrivare: Från engelska motor controller eller electronic speed control (ESC). En komponent som utifrån styrsignaler driver motorns riktning och hastighet. Motordrivarna omvandlar likspänning från en kraftkälla till tre stycken fyrkantsvågor för att förse motorerna med ström.

Borstlösa motorer: Elektriska motorer med en roterande permanentmagnet och stationära spolar vilket reducerar mekaniskt slitage.

Quadcopterns koordinatsystem: Quadcopterns kordinatsystem är samma som det klassika kartesiska koordinatsystemet. X-axeln utgörs av quadcopterns riktning framåt, dess längsaxel. Y-axeln utgörs av den axel som befinner sig i samma plan som dess x-axel men vinkelrät mot denna. Quadcopterns y-axel kan även benämnas tvärsaxel. Z-axlen är den axel som är vinkelrät mot både x- och y-axeln. Det är den axel quadcoptern rör sig längs då den endast förflyttar sig i höjdled.

Roll: Ordet roll är gemensamt för både svenskan och engelskan och betyder rotationsrörelse kring längdaxel. Rotationsrörelsen är alltså kring quadcopterns x-axel.

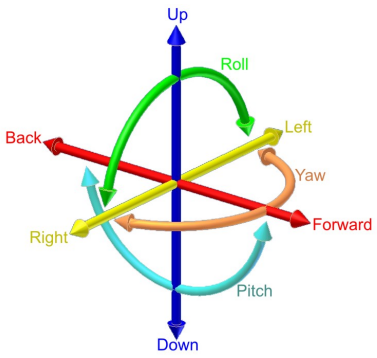
Aileron: En av signalerna från RF-mottagaren som ger upphov till roll.

Pitch: Ordet pitch definieras som rotationsrörelse kring tvärsaxel. Rotationsrörelsen är alltså kring quadcopterns y-axel.

Elevator: En av signalerna från RF-mottagaren som skapar rotation kring y-axeln.

Yaw: Yaw används på både svenska och engelska och motsvarar den återstående rotationsrörelsen vilket är den rotation som sker i horisontalplanet kring z-axeln, se Figur 1.1

Rudder: En av signalerna från RF-mottagaren som skapar rotation kring z-axeln.



Figur 1.1: De olika rörelse- och rotationsriktningarna för quadcoptern [51].

Raspberry Pi: En enkortsdator med linuxbaserat operativsystem lagrat på ett minneskort av typen microSD. Raspberry Pi:n är försedd med olika ingångar så som USB, tcp/ip och HDMI. Dessutom tillhandahåller den ett antal in- och utpinnar för att hantera exempelvis sensorer.

Pipe: Pipes är en typ av interprocesskommunikation och är en metod för att kommunicera mellan två eller flera parallella processer som körs samtidigt på ett operativsystem [43].

BEC: Förkortning av den engelska termen Battery Eliminator Circuit, batterieliminatorkrets. BEC:n omvandlar batterispänning till lägre spänning, som i det här fallet ersätter behovet av ett extern batteri till 5V-matningarna i systemet.

GPS: GPS står för Global Positioning System och är ett globalt system för beräkning av position med hjälp av satelliter.

RTK: RTK står för Real Time Kinematic och betyder relativ bärvågsmätning i realtid. RTK används för kompensering av mätfel vid GPS-mätningar.

1.3 Syfte

Syftet med projektet var att utveckla en plattform för autonom styrning av en quadcopter. Tanken med plattformen var att skapa förutsättningar för att utveckla autonoma funktioner som start, landning och navigering. Dessutom skulle möjligheten att utveckla fler autonoma funktioner som exempelvis hinderdetektion samt inhämtning och vidarebefordring av olika typer av information främjas.

1.4 Mål

På hårdvarusidan var målet att plattformen, innehållande en quadcopter, skulle utrustas med nödvändiga komponenter för såväl manuell som autonom styrning. Det skulle även implementeras ett säkerhetssystem som möjliggör att en operatör kan övergå till manuell styrning vid problem med de autonoma funktionerna.

På mjukvarusidan var målet att plattformen skulle innehålla drivrutiner för de implementerade komponenterna, bearbetning av insamlad mätdata, strukturer för dataflöde samt ett gränssnitt för styrning av quadcoptern.

Det slutgiltiga målet för plattformen var att påvisa att reglering av respektive frihetsgrad var möjligt.

1.5 Avgränsningar

Förutsättningarna för projektet var att en manuellt manövrerbar quadcopter, med möjlighet till vidareutveckling, skulle tillhandahållas.

En begränsad budget innebar att avvägningar mellan pris och funktion fick göras vid val av komponenter. Detta innebar att det i vissa fall fick göras avkall på komponenternas prestanda vilket skulle kunna göra sig till känna vid flygning i svåra situationer där hög precision krävs.

På grund av projektets tidsram spenderades ingen tid på att förse quadcoptern med någon form av väderskydd och därmed togs ingen hänsyn till väderförhållanden.

1.6 Arbetsmetod

Arbetet var ett utvecklingsprojekt vilket innebar att idéer och tankar kring arbetsgången föddes allt eftersom projektet fortlöpte. För att i efterhand generalisera arbetes olika delar delades det upp i ett antal olika områden som berörts. Till en början skapades en gemensam bild över projektets mål och möjliga helhetslösningar diskuterades. Tillhandahållet materiel utvärderades och nya komponenter valdes och köptes in. Dessa implementerades för att vidare funktionstestas. Allt större tester genomfördes, där fler och fler komponenter var involverade. Rapportens struktur följer arbetets gång i kronologisk ordning.

2
Teori

För att erhålla önskat resultat har olika teoriområden undersökts. I det här kapitlet förklaras projektets teoretiska bakgrund till dess huvudkomponeter mer i detalj.

2.1 Pulsbreddsmodulering

Pulsbreddsmodulering (PWM) är en metod där man delar in den tillförda effekten i cykeltider. Cykeltiden är en kort tid i förhållande till den anslutna objektens tidskonstanter. Under varje cykeltid styr man den tillförda effektens tillslagstid, det vill säga hur stor del av varje cykeltid ska den tillförda spänningen vara på. Den pålagda spänningen är alltså konstant och hur mycket effekt som tillförs bestäms av tillslagstiden.

Då cykeltiden är kort i förhållande till objektets tidskonstanter kommer objektet inte kunna urskilja de snabba förändringarna av tillslagstiden. Objektet kommer därför endast att se medelvärdet av spänningen som i sin tur ger upphov till effektutvecklingen. Förhållandet mellan pulslängden (tillslagstiden) och periodtiden (cykeltiden) uttrycks i procent och kallas pulskvot eller, på engelska, duty cycle. [5]

Pulsberddsmodulering kan även användas vid informationsöverföring. Mottagaren får då tolka signalens karaktäristik för att utläsa informationen som överförs.

2.2 Flygkontroller

En flygkontroller har med ett antal inbyggda sensorer uppgiften att omvandla signalerna från en RF-mottagare till en drivsignal för motordrivare på en multirotor beroende på flygkontrollerns tillstånd. En flygkontroller innehåller åtminstone ett gyroskop som tillsammans med algoritmer används för att reglera hastighet och möjliggöra stabil flygning [1]. Signalerna som hanteras i flygkontrollers är PWM-signaler.

Insignalerna till en flygkontroller är vid manuell styrning signaler från en RF-mottagare. Oftast mottas en signal för rotation kring varje axel och en för förflyttning i höjdled. Flygkontrollern omvandlar insignalerna beroende på tillståndet, styr spänningen till de individuella motordrivarna, och kan på så sätt framkalla önskad rörelse hos quadcoptern.

2.3 Motordrivare

En motordrivare har till uppgift att utifrån styrsignaler från flygkontrollern ställa omkopplingsfrekvensen av fälteffekttransistorer för att förse bortslösa motorer med en motsvarande AC-signal. Borstlösa DC-motorer kräver mindre underhåll eftersom de har färre delar som utsätts för slitage än traditionella DC-motorer [2].

I motordrivaren finns även en BEC som vanligtvis spänningsförsörjer RF-mottagare och andra komponenter i närheten av motordrivaren med lägre spänning än batterispänningen [3].

Generellt brukar motordrivare acceptera en 50 Hz PWM-signal som insignal med en pulsbredd från 5% till 10%, där 5% motsvarar 0 % pulsbredd av maximalt gaspådrag och 10% pulsbredd motsvarar 100% av maximalt gaspådrag. [4]

2.4 Raspberry Pi

Raspberry Pi 2 modell B är en dator som utvecklats av det brittiska företaget Raspberry Pi Foundation. Datorn har en 900 MHz quadcore ARM Cortex-A7 processor och 1GB RAM. Hårdvarumässigt har Raspberry Pi Model B dessutom 4 USB-portar, 40 GPIO-pinnar, HDMI-port, ethernet-port och kortplats för micro SD [6].

För att kommunicera med externa enheter finns GPIO-pinnar som arbetar med 3.3 V logik, samt flertalet olika bussar. En av bussarna är I^2C-bussen som är en dubbelriktad synkron seriell kommunikationsbuss och kan används av exempelvis sensorer som kommunikationsport. [7]

Eftersom Raspberry Pi 2 modell B har en ARMv7 processor är det möjligt att köra ARM GNU/Linux-distributioner, som Snappy Ubuntu Core och Windows 10. Andra exempel på operativsystem som går att köra på en Raspberry Pi är RASPBIAN, PIDORA, RISC och operativsystem med fokus på mediacenter så som OPENELEC och RASPBMC [19].

2.5 GPS

Idag används satellitpositionering, GNSS (Global Navigation Satellite Systems) i stor utsträckning för positionering på jordytan. Det finns flera olika system som ingår i GNSS, däribland GPS, GLONASS och Galileo, där GPS är det mest kända [9]. GNSS är användbart då det är möjligt att få en positionsangivelse på hela jordytan [9]. Dock har positionerna felvärden i storleksordningen 5-15 meter [10] vilket beror på faktorer såsom störningar i atmosfären och så kallade klockfel [15].

Det finns flera system för att förbättra noggrannheten hos GNSS mätningar som till exempel DGPS (differentiell GPS) [14] och RTK (relativ bärvågsmätning i realtid)

[12].

2.6 Nätverks-RTK

RTK används för att förbättra precisionen för en positionsbestämning från GPS:en. Vid användning av RTK kan användaren etablera en referensstation för att kompensera för de fel som nämns i Avsnitt 2.5 [12]. Referensstationen består av en fast GPS-mottagare där man i förväg noggrant mätt ut den verkliga positionen [12]. Genom att ta emot positionsdata från satelliter kan man avgöra hur stort fel de olika satelliterna rapporterar, och delge denna data till användaren som i sin tur kan kompensera för detta i positionsbestämningen [12]. Detta system fungerar alltså endast i närheten av den aktuella referensstationen, där de atmosfäriska förhållandena är approximativt desamma för GPS-mottagare såväl som referensstation [12]. Uppkoppling mot en referensstation kallas enkelstations-RTK [13].

Denna teknik har vidareutvecklats till så kallat nätverks-RTK där permanenta referensstationer placerats ut för att täcka stora områden [13]. Nätverks-RTK möjliggör system där data från flera stationer ger bättre korrektionsdata än en ensam station[13]. Med nätverks-RTK samverkar ett antal permanenta referensstationer som kommunicerar via en driftledningscentral som i sin tur förser användaren med korrektioner. I Sverige finns ett nätverks-RTK med namnet SWEPOS som tillhandahålls av Lantmäteriet [13]. Denna tjänst är gratis för studie- och forskningsändamål.

Den vanligaste typen av nätverks-RTK kallas VRS (virtuell referensstation) [13]. Tekniken bygger på att driftledningscentralen skapar en virtuell referensstation i nära anslutning till användarens utrustning [13]. Detta liknar det tidigare fallet med en enkelstations-RTK, med skillnaden att nätverks-RTK kräver tvåvägskommunikation av användarens utrustning. Användaren måste dels rapportera sin ungefärliga position, och dels kunna mottaga korrigeringsdata [13].

Mätosäkerheten för nätverks-RTK-tekniken är i storleksordningen av 1-2 centimeter i planet och 2-3 centimeter i höjdled om avståndet mellan referensstationerna inte är mer än 60-70 kilometer, vilket är ett typsikt avstånd i Sverige. [13]

2.7 Operationsförstärkare

En operationsförstärkare, förkortat OP eller OP-amp, är en komponent som förstärker spänningsskillnaden mellan dess ingångar. Förstärkningen är hög och vid analytiska beräkningar används ofta en så kallad ideal OP som modell där förstärkningen antas vara oändlig. Dessutom görs antagandet att OP:ns inimpedans likaså är oändlig och att dess utimpedans är noll. I verkligheten är så icke fallet. Förstärkningen för operationsförstärkare LM358 är exempelvis 100 dB [27] och generellt för en OP är inimpedansen 2-6 Mohm och utimpedansen ungefär 50 ohm [28].

OP:n har hög inimpedans vilket medför att dess påverkan på den övriga kretsen minimeras. På motsvarande sätt har OP:n låg utimpedans för att kretsen den förser med ström ska hålla en konstant spänning oavsett ström, det vill säga det inre spänningsfallet i OP:n minimeras [29].

Då OP:n förses med olika typer av återkoppling kan OP:n användas till olika funktioner. Ett par exempel på vanliga kopplingar är förstärkare och differentialförstärkare [30].

2.8 Filter

Ett filter är en elektrisk krets eller algoritm som släpper igenom signaler med vissa frekvenser men dämpar signaler med andra frekvenser [26]. Det frekvensområde där signaler släpps igenom kallas passband och det frekvensområde där signaler dämpas kallas spärrband. [26]

Filtrets pass- respektive spärrband klassificeras utifrån var i frekvensplanet de ligger. Uppdelningen brukar ske i lågpassfilter, högpassfilter, bandpassfilter och bandspärrfilter. Ett idelat lågpassfilter släpper igenom frekvenser upp till en så kallad gränsfrekvens samtidigt som det spärrar för frekvenser som är högre än gränsfrekvensen. De andra filtren fungerar på motsvarande sätt [26].

Beroende på filtrets ingående delar kan de även delas upp i passiva, aktiva eller digitala filter. Ett passivt filter kan innehålla passiva komponenter så som resistorer, induktorer och kondensatorer. Ett aktivt filter kan innehålla aktiva komponenter, det vill säga komponenter som kräver spänningsmatning för att bearbeta signalen, så som förstärkare. Digitala filter filtrerar digitala signaler i en dator eller signalprocesssor [26]. Ett exempel på ett digitalt filter är ett glidande medelvärde, som är en metod för att bilda en serie av medelvärden utifrån en viss mätdata. Varje värde ut ur filtret är ett medelvärde av ett visst antal ofiltrerade värden, och bidrar därför till att släta ut högfrekventa beteenden. [50]

2.9 Sensorer

Den grundläggande funktionen för ett antal sensorer beskrivs övergripande nedan.

Ultraljudssensor

En ultraljudssensor används för att bestämma avståndet mellan sensorn och ett solitt föremål. Detta genom att en givare sänder ultraljudspulser som kan reflekteras mot solida föremål. En mottagare på sensorn kan känna av en sådan puls som reflekterats. En kontrollkrets på sensorn kontrollerar sändning och mottagning av dessa pulser, och genom att mäta tiden som pulsens färdas kan avståndet till föremålet bestämmas. Ultraljudsensorn HC-SR04 inkluderar givare, mottagare och kontrollkrets [31]. Sensorn ansluts till konstant spänning, jord och två stycken signalkablar, *Trig*

och *Echo* (in- resp. utsignaler). En mätning påbörjas genom att ge en 10 ms signal till *Trig*-porten, varefter kontrollkretsen samordnar ett antal ultraljudspulser som sänds från sensorn. Dessa ljudpulser kan studsa tillbaka mot sensorn ifall de stöter på ett solitt föremål, vilket den inbyggda mottagaren kan registrera. Kontrollkretsen skickar i detta fall tillbaka en hög puls på *Echo*-porten. Tiden som denna signal är hög motsvarar tiden som ljudpulsen färdats i luften [32]. Genom att använda kunskap om ljudets hastighet i luft, $v_l \approx 340m/s$ vid 15° [33] kan avstånd till detekterat föremål beräknas enligt

$$s = t_h \frac{v_l}{2} = 170t_h \tag{2.1}$$

där s är avståndet i meter och t_h är den höga pulsens varaktighet på *Echo*-porten i sekunder [32].

Barometer

En barometer är ett instrument för mätning av lufttryck. Lufttryck mäts i hektopascal (hPa) och minskar med ca 1 hPa var åttonde meter över havsnivån, och kan därför användas till att bestämma absolut höjd över havsnivå. Lufttrycket kan dock variera dagligen mellan 950 hPa och 1050 hPa, vilket skapar behov av en noggrann mätning av lufttryck vid havsnivå för att en absolut höjd ska kunna bestämmas. Lufttrycket varierar också med lufttemperatur, där en lägre temperatur leder till snabbare avtagande lufttryck med höjd [34]. Om lufttrycket beräknats med hänsyn till lufttemperatur kan den absoluta höjden beräknas enligt

$$h = 44330(1 - (\frac{P}{P_0})^{\frac{1}{5.225}}) \tag{2.2}$$

där h är höjden i meter, P är uppmätt tryck och P_0 trycket vid havsnivå. BMP180 är en digital barometer tillverkad av Bosch Sensortec som mäter både tryck och temperatur som används för att bestämma lufttrycket i barometerns närhet. Barometern ansluts till spänning, jord och en I^2C-buss. En given algoritm används för att hämta information om aktuellt lufttryck och temperatur utifrån ett tiotal variabler från flera register på barometern. Denna process beskrivs i sensorns datablad [38].

Magnetometer

En magnetometer är en sensor för mätningar av magnetiska fält och kan användas likt en kompass för att avgöra dess riktning i förhållande till de magnetiska polerna [35]. HMC5883L är en 3-axlig magnetometer tillverkad av Honeywell. Att den är 3-axlig innebär att magnetfältsstyrka mäts i tre axlar, X, Y, Z i ett vänsterorienterat koordinatsystem. Om X och Y är komponsanterna i horisontalplanet, där X är huvudriktningen, och Z är vertikalkomposanten så kan magnetometerns orientering i förhållande till omgivande magnetfält beräknas. Under förutsättning att jordens magnetfält är det enda inom räckhåll för magnetometern, och att den ligger plant i X/Y-planet så kan orienteringen beräknas enligt

$$Direction(y > 0) = 90 - \frac{\arctan \frac{x}{y} * 180}{\pi} \tag{2.3}$$

$$Direction(y < 0) = 270 - \frac{\arctan\frac{x}{y} * 180}{\pi} \tag{2.4}$$

$$Direction(y = 0, x < 0) = 180.0 \tag{2.5}$$

$$Direction(y = 0, x > 0) = 0.0 \tag{2.6}$$

där rakt nordlig orientering ger $Direction = 0°$ på en 360° gradskiva. [45]

3
Utgångspunkt

För att uppnå målen utarbetades tankar och idéer kring en komplett fungerande helhetslösning som presenteras i kapitlet. Utifrån dessa tankar och idéer skapades en budget över vad som behövde införskaffas, där befintligt materiel togs i beaktning.

3.1 Tillhandahållet materiel

Vid projektets start var en grundförutsättning att det skulle finnas tillgång till en fungerande quadcopter. Quadcoptern togs över från ett tidigare års kandidatarbete av Andreas Andersson, Christopher Svensson, Daniel Pihlquist, Joakim Larsson och Mattias Wasteby [52] och grundtanken var att endast byta ut flygkontrollern då tidigare grupp själva programmerat den. Det var dock nödvändigt att testa resterande hårdvara grundligt för att avgöra vad som kunde användas, vad som behövde uppdateras och vad som behövde bytas ut. Tidiga tester visade att motordrivarna var gamla och ouppdaterade. Detta medförde problem i form av att motorerna vid låga varvtal stannade och sedan inte startade igen vid ökat gaspådrag. Under flygning får det här scenariot givetvis aldrig äga rum. Problemet kunde eventuellt lösas genom uppdatering av motordrivarna alternativt införskaffa nya. För uppdatering krävdes inköp av ett programmeringsverktyg. Då gruppen hade tillgång till egenägd hårdvara kunde jämförelser med en annan typ av motordrivare enkelt göras. Det visade sig att de egenägda motordrivarna var både stabilare och gav snabbare respons. Eftersom en uppdatering av de gamla motordrivarna inte garanterade tillräcklig förbättring gjordes därför valet att köpa in nya istället för att köpa in ett programmeringsverktyg.

3.2 Lösningsidéer

De delfunktioner som systemet ska innehålla presenteras nedan. En översiktsbild över det tänkta systemet presenteras i Figur 3.1.

Beräkningsnod

För att beräkna sensordata och skicka styrsignaler inom plattformen behövs en beräkningsnod. Vikt och prestanda är dom avgörande faktorerna vilket resulterade i ett val mellan en Arduino och en Raspberry Pi. En Raspberry Pi är mindre anpassad för sensorer, men går ändå att programmera för detta ändamål. Dock har Raspberry

Pi:n en mer beräkningskraftig processor vilket övervägde kompabiliteten för sensorer då flertalet processer och sensorer ska kunna användas samtidigt.

Kommunikation

För att plattformen ska kunna initiera program och få användarinput måste den kommunicera med en extern dator. För att uppnå detta finns ett flertal lösningar och i plattformen har stöd för 3G-modem och WiFi implementerats. Under projektet användes WiFi på grund av flygning inomhus. Som en säkerhetsåtgärd och testningsverktyg användes en RF-signal som kontrollerades manuellt av en operatör.

Navigation

Quadcopterns position bestäms via GPS. GPS:en är ansluten till SWEPOS RTK-tjänst via RTKLIB och med hjälp av detta korrigeras GPS-positionen för att en högre precision ska uppnås. I höjdled används en kombination av GPS, barometer och ultraljudssensor för att bestämma avstånd till marken. Att enbart utgå ifrån GPS-bestämning av höjd är riskabelt i de fall GPS:en inte har tillräckligt bra satellitmottagning, eller då internetuppkopplingen är för svag för att SWEPOS RTK-tjänst ska fungera väl. Ultraljudssensorn är riktad ned mot marken och är väldigt precis men har begränsad räckvidd, varför en barometer används för att komplettera höjdbestämning på högre höjder.

Mjukvara

All mjukvara körs inne på Raspberry Pi:n. Ett program har övergripande ansvar för att ta emot sensordata, användarinput och mata ut styrsignaler till flygkontrollern. Detta program är huvudprogrammet på plattformen och det program som alla komponenter kommunicerar med. Detta program innehåller även ett gränssnitt för kommunikation med användaren, och innehåller funktioner för styrning, loggning och flygning.

Säkerhetssystem

För att kunna hantera en situation där fel i hård- eller mjukvara i Raspberry Pi:n uppstår implementerades ett säkerhetssystem. Systemet inkluderar en RF-sändare och RF-mottagare samt en multiplexer med tillhörande krets, och möjliggör att en operatör kan överta kontrollen av quadcoptern och styra denna manuellt vid behov. Systemet aktiveras genom att operatören skickar en signal från RF-sändaren, som mottagaren skickar vidare till kretsen innehållande en multiplexer. Kretsen ger en styrsignal till multiplexern som i sin tur blockerar alla signaler från Raspberry Pi:n och istället släpper igenom kommandon från RF-sändaren. Multiplexerns tillhörande krets utför en jämförelse av de två signalerna genererade av de två olika lägena på RF-sändaren för att utifrån det kunna avgöra från vilken källa flykontrollern ska mottaga sina signaler. För att säkerhetssystemet ska fungera krävs dock att quadcoptern är inom synhåll för operatören, inom räckvidd för RF-sändaren samt

att operatören är uppmärksam och kan avgöra när övertagande av styrningen är nödvändigt.

För att skapa en extra säkerhet har även en voltmätare implementerats på quadcoptern. Denna komponent är direkt kopplad till batteriet och mäter batterispänningen. När batteriet börjar ta slut ljuder ett larm och systemet kräver därmed att operatören är inom hörhåll så att styrningen kan tas över manuellt för att en säker landning ska kunna utföras.

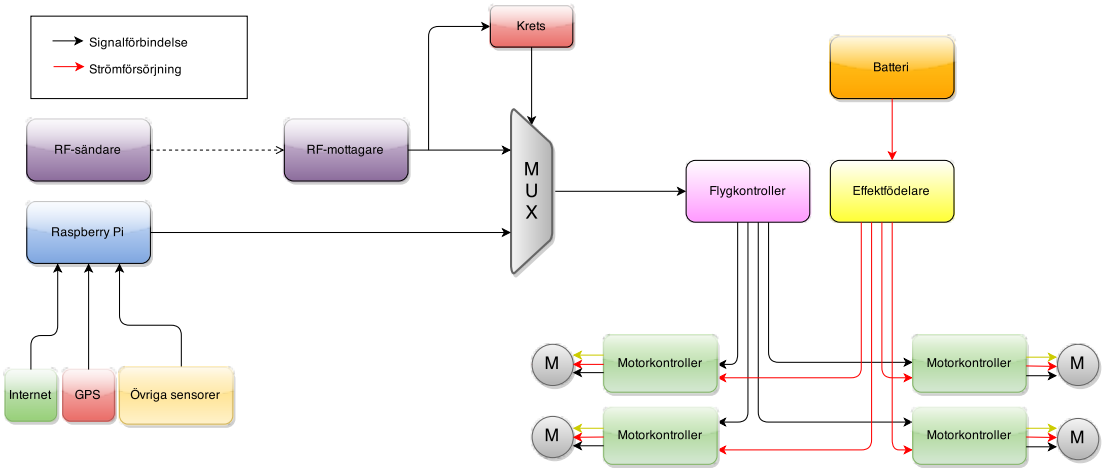
Strömförsörjning

Quadcoptern förses med energi från batteriet som är beläget centrerat på quadcoptern. Batteriet matar både en strömdistributör som fördelar strömmen till de fyra motordrivarna samt en spänningsregulator som spänningsförsörjer Raspberry Pi:n med 5 V. De övriga 5V-matade komponenterna, flykontrollern och multiplexern, matas från den interna batterieliminatorn, i en utav motordrivarna. Multiplexern kräver dessutom batterispänning. De USB-anslutna komponenterna, internetdongeln och GPS:en, förses med ström via Raspberry Pi:ns USB-uttag.

Spänningsregulatorn implementerades i efterhand och var från början inte planerad. Tidigare försågs 5V-matningarna med spänning från en utav motordrivarnas inbyggda batterieliminator. Vid den första driftsättningen av quadcoptern visade det sig att denna motordrivare närmade sig en kritisk temperatur och klarade alltså inte att leverera tillräckligt stor ström. Tanken var då att parallellkoppla 5V-matningarna från de fyra motordrivarna för att på så sätt minimera den enskilda strömen som de var och en måste leverera. Efter rådfrågning av kunniga och insatta människor [20] inom området erhölls istället tipset att använda en spänningsregulator då det i annat fall skulle kunna uppstå icke önskvärda resononasföreteelser. All 5V-matning försörjdes därför av spänningregulatorn. Vidare kom detta att ändras så att endast Raspberry Pi:n försörjdes från spänningsregulatorn. Detta för att eventuellt erhålla möjligheten att överta kontrollen vid allvarliga fel på Raspberry Pi:n.

3.3 Budget

På grund av att quadcoptern som tillhandahölls från tidigare kandidatgrupp var bristfällig behövde ett antal komponenter införskaffas. De komponenter som införskaffades för att uppnå de tänkta grundföruttsättningarna var flygkontroller, motordrivare och RF-mottagare. För att utveckla plattformen och möjliggöra autonom styrning krävdes inköp av ytterligare komponenter. Dessa var Raspberry Pi, minneskort, GPS, antenn och sma-kontakt, sensorer samt multiplexer med tillhörande komponenter. I Tabell 3.1 åskådliggörs inköpta komponenter med tillhörande priser. Precis som Tabell 3.1 visar översteg den totala summan av inhandlade komponenter det budgeterade beloppet. Detta beror delvis på att budgeten gjordes i ett tidigt skede och att arbetsgången och nödvändiga komponenter inte var klart vid denna



Figur 3.1: Flödesschema för systemet

tid. Vid tidpunkten av budgetens skapande var den inhämtade informationen otillräcklig vilket resulterade i att en del budgeterade komponenter krävde ytterligare tillbehör, så som ett minneskort till Raspberry Pi:n och en antenn av typen Navilock NL-280GG till GPS:en. På grund av att priset togs i beaktning vid val av komponenter kunde kostnaderna per komponent hållas under den övre budgeterade gränsen med marginal och det fanns därmed rum för dessa oväntade nödvändiga tillbehör. Ytterligare en anledning till att budgetens totala övre gräns överskreds är att Raspberry Pi:n överhettades på grund av en kortslutning och behövde bytas ut. Trots oväntade kostnader på 1109 SEK överskreds budgeten totalt endast med 375 SEK, vilket ungefär motsvarar den oväntade utgiften för Raspberry Pi:n.

3.4 Design

Då quadcoptern tillhandahölls från en tidigare kandidatgrupp omfattade designen endast placering av komponenter då dessa var fler och annorlunda i det här projektet. Tanken var att, i likhet med tidigare års kandidatgrupp, designa och 3D-printa ett komponenthus. Denna tanke avstyrdes då antal nödvändiga komponenter samt dess geometrier inte var känt. Som följd gjordes provisoriska lösningar under tiden med hjälp av skruvar, buntband, gängade stavar och liknande. Långa sladdar snurrades runt olika fasta förmål samt fästes med buntband för att säkerställa att de aldrig hamnar i vägen för något av rotorbladen. Fastsättningen av komponenterna förbättrades under tiden projektet fortlöpte och blev tillslut en permanent lösning. Detta val gjordes på grund av att lösningen medförde en kompakt placering av komponenterna som i sin tur medförde en stabil quadcopter. Ett komponenthus skulle inte tillåta en lika tät placering av komponenterna och huset i sig skulle även öka quadcopterns vikt vilket är ytterligare en nackdel. Typen av fästen som används skapar dock problem vid montering och demontering vid felsökning av komponenterna. Många komponenter kräver att fästen klipps av och därefter krävs nytt materiel vid

Tabell 3.1: Projektets ingående komponenter och budgeterade priser.

Komponenter	**Pris (SEK)**	**Budgeterat pris (SEK)**	**Kvar efter avslutat projekt**	
Ram	-	-	Ja	*
Motorer	-	-	Ja	*
Rotorblad	-	-	Ja	*
Motordrivare	460	-	Ja	
Flygkontroller	190	200 - 250	Ja	
RF-sändare	-	-	Nej	**
RF-mottagare	130	-	Ja	
Magnetometer	50	50 - 100	Ja	
Barometer	50	-	Ja	
Ultraljudssensor	59	50 - 100	Ja	
Spänningsregulator	-	-	Ja	***
Batteri	-	-	Ja	*
Voltmätare	-	-	Nej	**
3G-modem	-	-	Nej	**
WiFi-dongel	-	-	Nej	**
GPS	558	500 - 1500	Ja	
Antenn m. kontakt	169	-	Ja	
Raspberrry Pi	399	300 - 500	-	
Ny Raspberry Pi	399	-	Ja	
Minneskort	170	-	Ja	
Komp. till multiplexer	191	-	Ja	
Totalt	**2825**	**1100 - 2450**		

* Komponenten erhölls från tidigare års kandidatarbete
** Komponenten fanns att tillgå inom gruppen
*** Donation

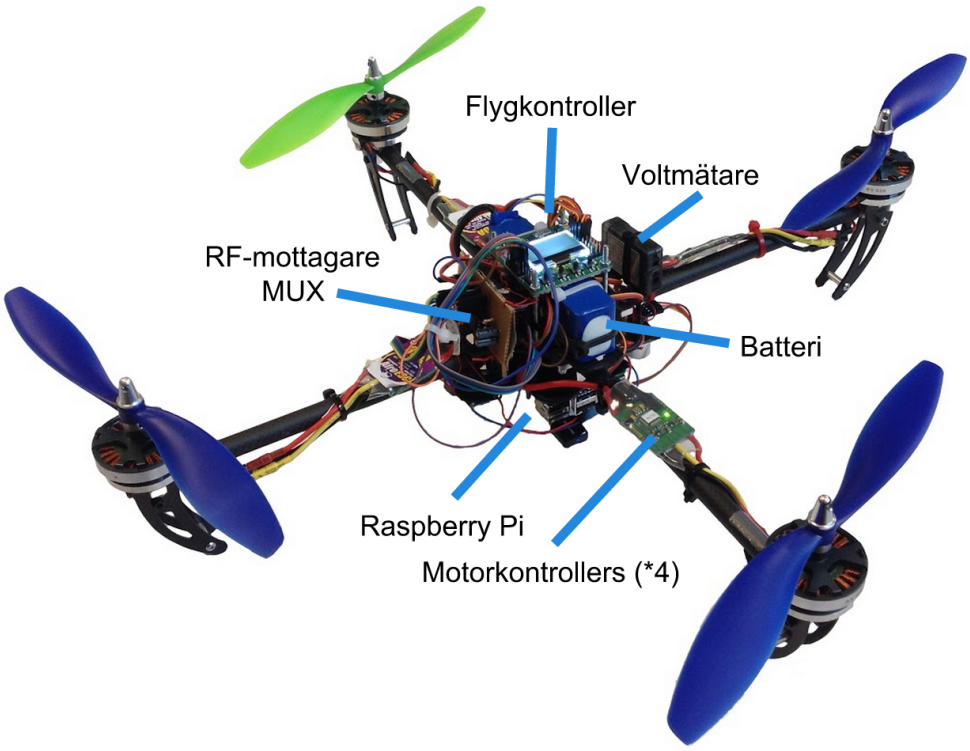
återfästning. Då produkten är färdig är detta inte ett problem men under arbete med och utveckling av quadcoptern skulle denna metod kunna förbättras och effektiviseras.

På grund av den stora mängden komponenter och den täta placeringen av dem har komponenterna isolerats med tejp på respektive baksida. Detta för att undvika kortslutning vid kontakt med metall på ramen eller andra komponenter.

Den slutgiltiga lösningen, Figur 3.2, av komponentplaceringen resulterade i att batteriet, på grund av dess tyngd, placerades centrerat direkt på quadcopterns stomme för att uppnå en mer stabil design vid flygning. Ovanför batteriet sitter flygkontollern. Vid ena långsidan sitter RF-mottagaren samt kretskortet med multiplexer och tillhörande komponenter. På andra sidan sitter voltmätaren. Resterande komponenter är placerade på quadcopterns undersida, med undantag för motordrivarna som sitter på respektive arm på ramen. Närmast mitten på undersidan sitter strömdistri-

butören. Under denna är Raspberry Pi:n placerad varpå barometer, magnetometer och GPS återfinns. På ena sidan om Raspberry Pi:n sitter ultraljudssensorn och på andra sidan är spänningsregulatorn placerad.

Vidare har det tagits hänsyn till de möjliga utvecklingar av projektet som en kamera kan resultera i. Då en lämplig placering av en kamera är på quadcopterns undersida finns det utrymme att placera denna i närheten av Raspberry Pi:n.



Figur 3.2: Översiktsbild av quadcopterns design och komponentplacering

4

Implementering av nya komponenter

Det här avsnittet tar upp de nya komponenter som införskaffats och som upptagit mycket resurser under projektet. Avsnittet beskriver också hur komponenterna modifierats och implementerats. Komponenterna är i huvudsak valda utifrån önskad funktion men även pris har tagits i beaktning. Att implementera komponenterna på ett bra sätt har lett till omfattande utmaningar.

4.1 Flygkontroller och motordrivare

Det tidigare kandidatarbetet hade som mål att designa en flygkontroller som via radiostyrning kunde reglera gaspådraget till motorerna och möjliggöra manuell styrning. Efter att förra årets kandidatgrupp[52] traderat nödvändig information angående dess erhållna resultat föddes insikten att deras utvecklade flygkontroller skulle vara svår att arbeta vidare med. Kommersiella flygkontroller är lätta att implementera och tillhörande öppen källkod finns ofta att tillgå[1]. Det som skiljer olika flygkontroller åt är i huvudsak dess olika färdigprogrammerade funktioner. En lista av de vanligaste funktionerna och önskade funktioner återfinns i Tabell 4.1. Då projektets ambition var att att skapa förutsättningar för att implementera en del av de färdiga funktionerna som återfanns hos vissa flygkontroller, uteslöts många modeller. Tidigare erfarenhet av flykontrollern KK2.1.5 från Hobbyking, dess utbud av funktioner, lätta integrering och pris resulterade i att denna köptes in.

Tabell 4.1: De önskade funktionerna av de tillgängliga hos olika flygkontroller.

Öppen källkod	Gyro-stabilisering	Själv-nivåreglering	Höjd-hållning	Position-hållning	Återvända hem	Viapunkts-navigering
☑	☑	☑	☐	☐	☐	☐

De olika funktionerna i Tabell 4.1 med beskrivning

- **Öppen källkod** tillåter alla användare att tillgå koden
- **Gyrostabilisering** använder ett gyroskop för att eliminiera oscilliationer
- **Självnivåreglering** använder en accelerometer och ett gyroskop för att låta quadcoptern återgå till horisontalplanet efter lutad flygning

- **Höjdnivåhållning** bibehåller quadcoptern på en konstant höjd med hjälp av en barometer
- **Positionshållning, återvända hem, viapunktsnavigering** använder alla en intern GPS för att genomföra olika uppgifter

Flygkontrollern blev tidigt implementerad på quadcoptern då den är en av de avgörande komponenterna för en fungerande mekatronisk lösning. Den nödvändiga 5V-matningen till flygkontrollern hämtades från BEC på motordrivarna. Från flygkontrollern hämtades därpå spänning till RF-mottagaren. Motordrivarna kopplades in till flykontrollerns utgångar. När inkopplingen var klar återstod en ledsagning genom ett grafisk användargränssnitt på flygkontrollern med tillhörande inställningsmenyer.

Quadcopterns design medför att flygkontrollern, som är rektangulär och nästan kvadratisk, lättast monteras så att flykontrollern följer samma placeringssätt som de övriga komponenterna, se Figur 3.2 i kapitel 3. På flygkontrollern finns en pil som markerar farkostens riktning framåt genom förhandsprogrammerade system. Pilen är markerad vid flygkontrollerns ena utkant, vinkelrät mot denna, och pekar utåt. På grund av flygkontrollerns placering pekar pilen längs med en av armarna på quadcoptern. Baserat på pilens riktning i förhållande till quadcopterns motorer bestäms i inställningsmenyn quadcopterns formation, som i detta fall är ordnad som ett plus. De två vanliga formationerna är plus respektive kryss vilket innebär att en respektive två motorer utgör riktningen framåt.

Internt i flygkontrollern finns en kaskadreglering där användaren kan ändra mellan att endast använda den ena interna PID-regulatorn eller båda under flygning. Vid aktivering av båda regulatorerna regleras quadcopterns lutning så att dess lutning motverkas och att quadcoptern återgår till ett horisontellt läge efter att quadcoptern har förflyttats i sidled, detta kallas också auto-level.

De nya motordrivarna, Afro ESC 30A [22], som valdes att ersätta de tillhandahållna motordrivarna fästes på quadcoptern. De nya motordrivarna matades liksom de gamla från strömdistributören. Vidare försörjde de motorerna med fyrkantsformade likspänningspulser och motordrivarnas signalkablar kopplades sedan in till flygkontrollern. Vilka motordrivare som ska kopplas till vilken port på flygkontrollern är närmare beskrivet i Appendix A.

4.2 Raspberry Pi

Quadcopterns beräkningsmässiga nav består av ett program som körs på en Raspberry Pi. Den fungerar som en nod där mätdata från de olika sensorerna tas emot, bearbetas och styrsignaler skickas vidare. Raspbery Pi:n levereras från fabrik utan varken minne eller operativsystem. Minnet köps som tillbehör i form av ett minneskort och operativsystemet får man själv ladda ned.

Ett operativsystem, Rasbian, laddades ned och lades in på minneskortet. Raspberry Pi innehåller till en början endast en hierarkisk mappstruktur med nödvändiga filer för att den ska kunna vara flexibel och anpassas efter användarens behov. Till Raspberry Pi:n laddas ändamålsenliga program och bibliotek ned för att tillgodose önskad funktionalitet.

Då Raspberry Pi:n inte har några förinstallerade program utöver vad som krävs för dess egna behov måste det till alla komponenter som ansluts installeras ett program som kommunicerar med denna. Även för att kommunicera med anslutningspinnarna på Raspberry Pi:n måste ett program installeras.

4.3 Internetuppkoppling

För att plattformen ska vara uppkopplad mot SWEPOS RTK-tjänst krävs internetuppkoppling. Detta löstes på två olika sätt, dels med ett 3G-modem, även kallat mobilmodem eller mobilt bredband, och dels med en WiFi-dongel. Att båda användes var delvis för att skapa flexibilitet under utvecklingsprocessen, där WiFi enkelt kan användas inomhus och ej medför dataförbrukning på 3G-modemet.

Ett 3G-modem fanns att tillgå inom gruppen och efterforskningar på nätet påvisade att modemet skulle fungera tillsammans med Raspberry Pi. Modemet låstes upp och en guide för hur man implementerar ett 3G-modem hittades på internet [55]. Guiden hänvisade till skriptet Sakis3g som är ett skript för uppkoppling via 3G-modem.

Nödvändig programvara vid namn Sakis3g installerades, och krävde enbart att en konfigueringsfil installerades för att uppkopplingen fortsättningsvis skulle ske automatiskt vid start av Raspberry Pi:n.

Utöver ett 3G-modem konfigurerades en WiFi-dongel till Raspberry Pi:n. Till denna fanns förinstallerad programvara där endast uppdatering av en konfigurationsfil krävdes.

4.4 GPS

Som GPS-mottagare valdes U-blox NEO-6T USB dongle. Förutom ett lågt pris uppfyller den kravet att kunna mata ut rådata för att kommunicera med RTKLIB. Inledningsvis testades GPS-mottagare och antenn på en extern Windows-dator. Detta för att lättare kunna anpassa inställningar och uppkoppling till nätverks-RTK med hjälp av det grafiska gränssnittet som inte fanns att tillgå på Raspberry Pi:n.

GPS med RTKLIB på Raspberry Pi

Programpaketet RTKLIB är skrivet i C vilket gör det möjligt att kompilera och köra på en Raspberry Pi. Filerna laddades ner från www.rtklib.com vilket är deras offi-

ciella hemsida [17]. För att utföra implementeringen användes en guide på internet, skriven av Rai Gohalwar [18].

I Windows-verisionen av RTKLIB kunde de valda inställningarna sparas ner till en konfiugrationsfil som senare kunde användas vid inställning av RTKLIB på Raspberry Pi:n.

Efter installationen av RTKLIB på Raspberry Pi:n innehöll konfigurationsfilen rtkrcv.conf specifika inställningar som utelämnats i den tidigare konfigurationsfilen deriverad från Windows-versionen av RTKLIB. Inställningarna i rtkrcv.conf var mer överordnade och innefattade exempelvis vilket protokoll som användes för GPS:en, vilken USB-port som denna var inkopplad i, sökväg för temporära filer och sökväg för uppstartfiler. Dessa inställningar anpassades efter uppkopplingen.

De tre uppstartfilerna ubx_raw_1hz.cmd, ubx_raw_5hz.cmd och ubx_raw_10hz.cmd kompletterades med uppstartskommandon i form av tal som tidigare modifierats enligt dokumentation på Emlids hemsida [54]. Filernas innehåll kan ses i Appendix C.

Baserat på den egengenererade konfigurationsfilen från Windows-versionen och den förinstallerade, skapades en ny konfigurationsfil med namn newopt.conf. Denna konfigrationsfil var en sammanställning av de två filerna och var den som senare användes.

Kommunikation med huvudprogram

På Raspberry Pi:n körs RTKLIB som ett fristående program, det vill säga att det startas separat från de övriga programmen. RTKLIB levereras visserligen med öppen källkod, men har flera metoder att leverera sin utdata till ett program som vill läsa av den. Det som valdes var en seriell utport /dev/ttyGPS, dit RTKLIB skriver data som sedan huvudprogrammet kan läsa av. /dev/ttyGPS är inte en fysisk port på Raspberry Pi:n, och kommer därför att raderas vid varje omstart av operativsystemet, varför den nu skapas som en del av initieringen i huvudprogrammet.

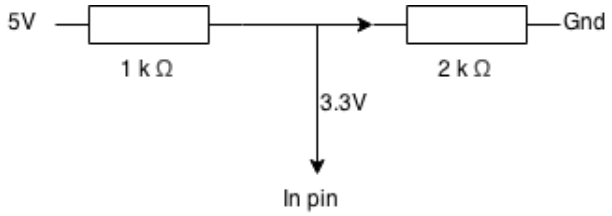
4.5 Sensorer

Sensorutrustningen består av en magnetometer, en ultraljudssensor samt en barometer. Magnetometern används för att avgöra quadcopterns bäring, barometern används som höjdgivare vid högre höjder än 4 meter, och ultraljudssensorn vid höjder lägre än 4 meter.

Ultraljudssensor

På låga höjder krävs hög precision och stor tillförlitlighet på mätdatan, därför används en ultraljudssensor. Den ultraljudssensor som köptes in till projektet är av

modell HC-SR04 och säljs ofta som tillbehör till Arduino. Sensorn mäter maximalt upp till 5 meter, med en mätosäkerhet på 3 millimeter [44].
Sensorn har en inspänning såväl som utspänning på 5V [24]. Då portarna till Raspberry Pi:n endast hanterar 3.3V [25] behövde denna sänkas, vilket gjordes med en så kallad spänningsdelning bestående av två resistanser enligt Figur 4.1.



Figur 4.1: Sänkt spänning på signalen från sensor

Sensorn är monterad på quadcopterns undersida, se Figur 4.2 och kopplad till Raspberry Pi:ns GPIO-pinnar enligt Figur A.1, Appendix A. Avläsningen av denna sensor sker i ett program som är döpt till HC-SR04.c, med funktionen *int getUltra()*. Funktionen skickar en begäran om mätning via *Trig*-porten och läser sedan på *Echo*-porten. Om *Echo*-porten ger en signal så sparas tiden som denna är hög och används för att beräkna avståndet enligt Ekvation 2.1, Avsnitt 2.9.

Barometer

En digital barometer av modell BMP180 inhandlades. Syftet med denna är att kunna mäta lufttrycket för att sedan beräkna en höjd då ultraljudssensorns räckvidd överskrids.

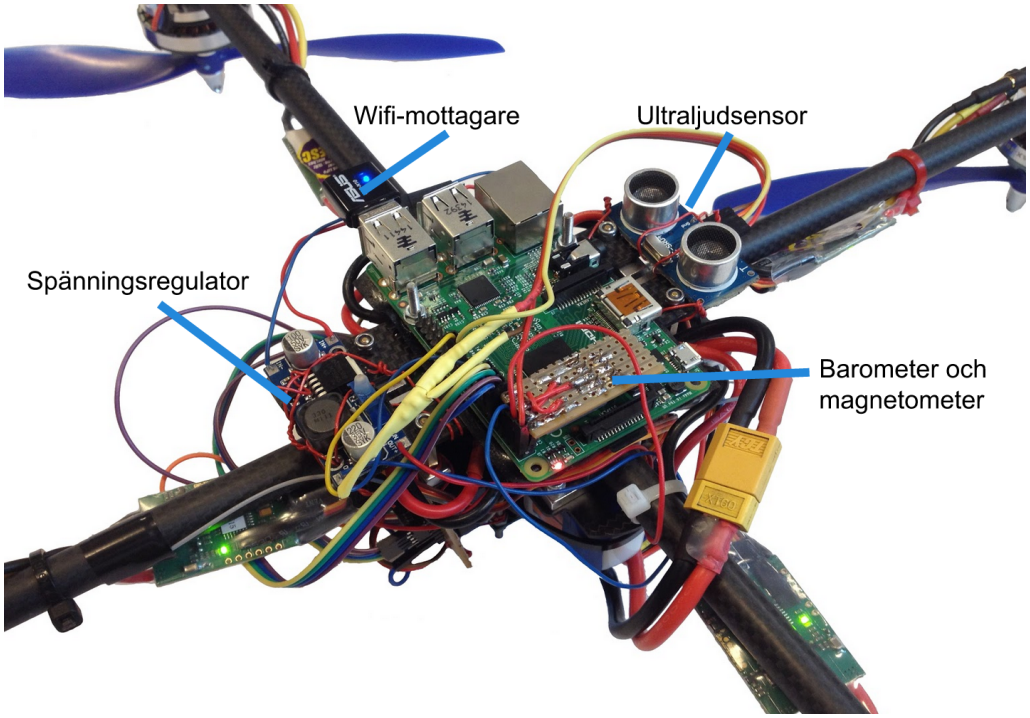
Barometern är monterad på ett kretskort på Raspberry Pi:ns GPIO-pinnar enligt Figur A.1, Appendix A, där den är kopplad till en I^2C-buss. Det finns färdigskrivna bibliotek för kommunikation mellan BMP180 och många populära plattformar såsom Raspberry Pi. Dock är detta bibliotek skrivet i Python [21], varför en egen drivrutin fick skrivas för implementation i C. Denna implementation utgick ifrån den algoritm som beskrivs i databladet för BMP180. [38] Höjden kunde sedan beräknas enligt Ekvation 2.2, Avsnitt 2.9.

Magnetometer

För att bestämma vilken bäring quadcoptern har används en magnetometer av modell HMC5883L från Honeywell. Det är en 3-axlig magnetometer som kommunicerar med andra komponenter via en I^2C-buss. På samma sätt som för barometern så finns det ett flertal färdiga bibliotek för denna sensor, men inte i C. Därför skrevs en egen drivrutin i HMC5883L.c.

Magnetometern är monterad på samma krets som barometern, på Raspberry Pi:ns GPIO-pinnar, där den är kopplad till samma I^2C-buss, på samma kretskort som barometern.

Från sensorn kan magnetfältsvärden ges i tre olika led, X,Y,Z. För att beräkna magnetometerns riktning används Ekvation 2.3-2.6, Avsnitt 2.9.



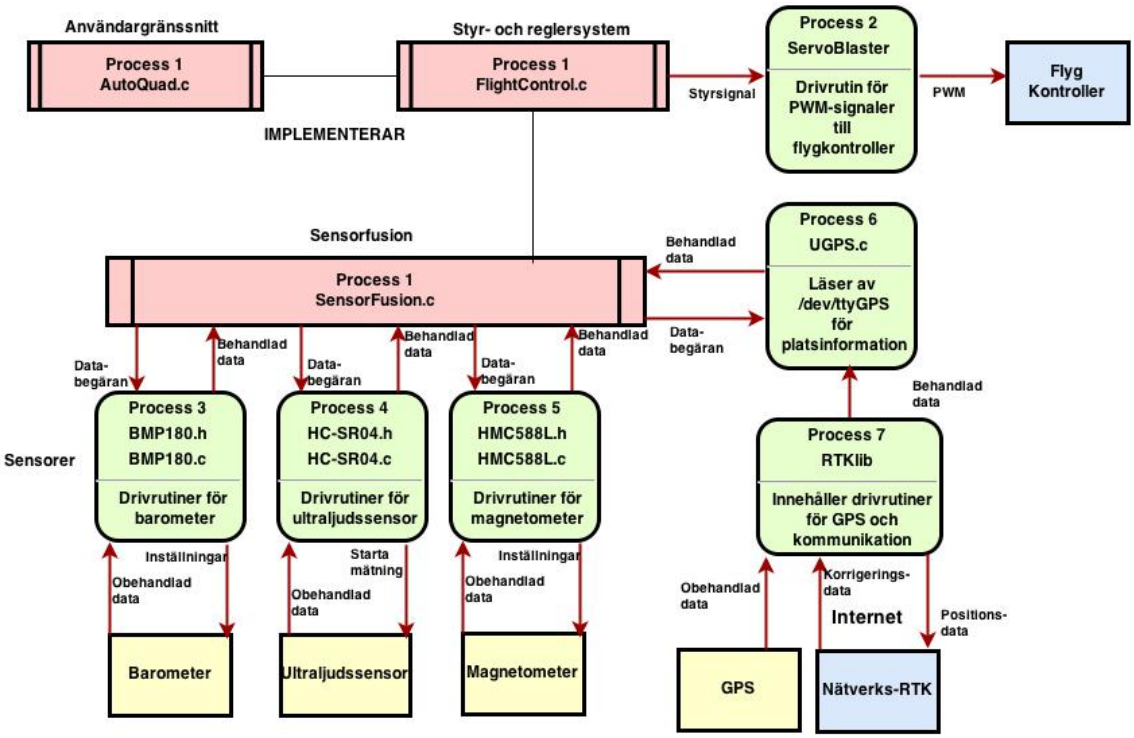
Figur 4.2: Quadcopterns undersida

4.6 Mjukvara

För att möjliggöra autonom styrning krävs grundläggande mjukvara och drivrutiner som sköter kommunikation mellan quadcopterns olika system och sensorer. För att uppnå detta har ett antal program från tredje part använts tillsammans med egenproducerad kod.

Plattformens mjukvara består av sju parallella processer, se Figur 4.3. Process 1 (AutoQuad) är huvudprocess och innehåller användargränssnitt (AutoQuad.c), sensorhantering (SensorFusion.c) och reglersystem (FlightControl.c). Process 2-6 är drivrutiner för kommunikation med flygkontroller, sensorer och GPS. Process 7 är programmet RTKLIB. Process 3-5 kommunicerar med process 1 via så kallade *pipes* som är en teknik för kommunikation mellan processer på samma operativsystem. En pipe bygger på att en fil öppnas samtidigt av två program, där det ena endast har skrivrättighet och det andra läsrättighet [53]. Process 7 matar ut data på den virtuella seriella porten /dev/ttyGPS vilken läses av process 6. Process 7 har även kommunikation med SWEPOS nätverks-RTK via internet.

Anledningen till uppdelningen i olika processer är att alla sensorer har olika maximal samplingshastighet. Barometern kräver dessutom en fördröjning under själva



Figur 4.3: Programhierarki som beskriver de olika nivåerna av programmen

mätningen. Med parallella processer kan varje sensor mäta med sin maximala samplingsfrekvens och behandla datan parallellt med att styrprogrammet körs. Styrprogrammet kan sedan begära uppdateringar i den takt de behövs. Parallella processer medför också att antalet felkällor hos huvudprogrammet minimeras, ifall en sensor slutar att fungera kan resten av programmet fortsätta köra.

De program och bibliotek från tredje part som har använts i projektet är följande:

- RTKLIB
- ServoBlaster
- WiringPi
- libconfig

RTKlib är ett program skrivit i C för berarbetning av GNSS data för förbättrad positionering. Programmet sköter all kommunikation med både GPS-modulen och SWEPOS. [17]

ServoBlaster är programet som sköter kommunikationen med flygkontrollern via PWM. För att kommunicera med SvervoBlaster görs anrop i terminalen där en pro-

centsats av maximalt utslag anges. [41]

WiringPi är ett bibliotek för C som underlättar användandet av Raspberry Pi:ns GPIO pinnar. I projektet används biblioteket för kommunikationen via I^2C och för att skapa fördröjningar i koden samt hålla koll på åtgången tid från programstart. [40]

libconfig är ett bibliotek för C och C++ som som läser in variabler från konfigurationsfiler [42]. En konfigurationsfil används för att inte viktiga variabler som kan behöva ändras inte ska behöva vara hårdkodad i källkoden. Detta underlättar för exempelvis testing av PID-parametrar.

Den programkod som är egenproducerad är skriven i programmspråket C. Bland de egna materialet finns även två bash script och ett konfigurationsfil. De egna materialet följer nedan.

- AutoQuad.c
- FlightControl.c
- SensorFusion.c
- BMP180.c
- HC-SR04.c
- HMC5883L.c
- UGPS.c
- testCases.c
- update.sh
- run.sh
- c.cfg

AutoQuad är plattformens huvudprogram som implementerar ett CLI (command-line interface) som tar hand om indata från användaren för att exekvera olika metoder som implementeras i andra filer. Programmet tar även hand om utdatan från andra funktioner och presenterar datan för användaren. AutoQuad inkluderar FlightControl.c, SensorFusion.c och testCases.c för att kunna exekvera deras funktioner. AutoQuad startar även de andra processerna som krävs för att systemen ska fungera.

FlightControl tar hand om komunikationen mellan Raspberry Pi:n och flygkontrollern. Filen innehåller funktioner för att ange värden från 0-100% på flygkontrollerns

ingångar via servoblaster. Det är även i denna fil framtida regleralgoritmer ska implementeras. I nuvarande version av FligtControl har en funktion implementerats som reglerar höjden av quadcoptern med hjälp av en PID-regulator.

SensorFusion innehåller funktioner för att sköta insamling av data från sensorerna samt GPS:en och göra den tillgänglig för de program som inkluderar SensorFusion.

BMP180 är en drivrutin som sköter kommunikationen med BMP180 sensorn via I^2C-bussen, och bearbetar rådata från barometern. BMP180.c får en databegäran av process 1 och placerar som svar data i en pipe ansluten till process 1.

HC-SR04 är en drivrutin för ultraljudssensorn som hämtar och bearbetar rådata från sensorn. HC-SR04.c får en databegäran av process 1 och placerar som svar data i en pipe ansluten till process 1.

HMC5883L är en drivrutin för magnetometern som hämtar och bearbetar rådata från sensorn. HMC588L.c får en databegäran av process 1 och placerar som svar data i en pipe ansluten till process 1.

UGPS läser av den virtuella seriella porten /dev/ttyGPS, där RTKLIB ger sin utdata. UGPS.c får en databegäran av process 1 och placerar som svar data i en pipe ansluten till process 1.

testCases är en fil som innehåller funktioner för testning av plattformen. Testerna används för att till exempel göra ett test där quadcoptern får ett stegökning av gaspådrag, och loggar sensordata under efterföljande tiden för på ett experimentellt sätt försöka förstå hur systemet beter sig.

update.sh är ett script som klonar den senaste versionen av projektet från GitHub [49] och kompilerar alla filer samt startar AutoQuad.

run.sh är ett script som startar AutoQuad.

c.cfg är en konfigurationsfil som innehåller inställningar för sensorerna samt parametrar för tester och höjdregleringen.

4.7 Multiplexer

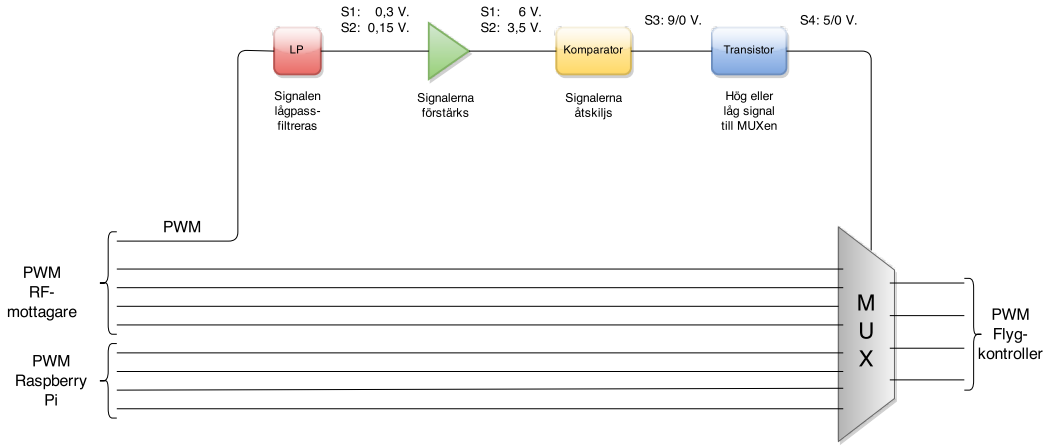
För att säkerställa att önskad funktionalitet erhålls och för att minimera felkällor valdes en analog krets för att urskilja autonom och manuell styrning av quadcoptern. Autonom styrning innebär här att flygkontrollern får insignaler från Raspberry Pi:n medan manuell styrning innebär att flygkontrollern får insignaler från

RF-mottagaren. RF-mottagaren har åtta kanaler varav quadcoptern endast använder fem (Roll, Pitch, Yaw, Throttle och en för auto-level), vilket medför att det finns tre kanaler att tillgå. En av de återstående kanalerna valdes att användas till att möjliggöra växling mellan autonom och manuell styrning.

Ändringen av autonomt respektive manuellt läge görs vanligtvis på distans varför det var ett naturligt val att använda en av kanalerna hos RF-mottagaren vilket erbjuder lång räckvidd. Alternativt skulle samma resultat kunna erhållas genom användning av internetuppkopplingen som också återfinns på quadcoptern. Detta alternativ valdes inte då växlingen mellan de två lägena bör finnas tillgängligt för operatören även om mjukvaruproblem uppstår.

Genom att använda en analog krets utesluts mjukvarufel vilket ökar kretsens tillförlitlighet. Ett alternativ hade varit att använda en mikrokontroller som exempelvis Arduino nano. På det viset hade större flexibilitet uppnåtts då den kunde använts för flera PWM-kanaler i framtiden. Men då kretsen endast ska urskilja en PWM-signal har säkerhet och tillförlitlighet prioriterats före flexibilitet som ändå inte är relevant i sammanhanget. Den implementerade kretsen skulle vid behov kunna designas om för att på så sätt krympa denna ytterligare. Viktskillnaden mellan kretsen och Arduino Nano är relativt liten och gör i detta fallet ingen skillnad.

Valet av ingångskälla till flygkontrollern sker med hjälp av en omkopplare på RF-kontrollen. Denna information överförs med en av de tre överflödiga kanalerna från RF-mottagaren. Signalen från RF-mottagaren är en PWM-signal med olika pulskvot beroende på läge på omkopplaren.



Figur 4.4: Förenklat flödesschema för multiplexern

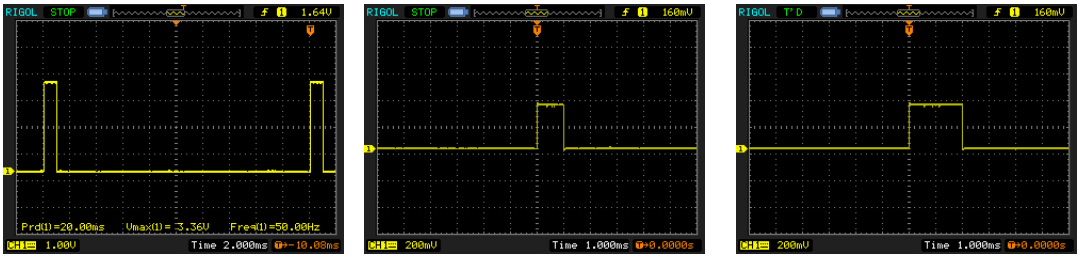
I Figur 4.4 ses ett schematisk flödesschema för kretsen. De två olika signalerna, omkopplaren i av- och på-läge, lågpassfiltreras för att skiljas från varandra. Den PWM-signal som har en liten pulskvot, det vill säga dess största del av perioden är noll, får en lägre spänningsnivå när denna lågpassfiltreras jämfört med PWM-

signalen med större pulskvot. De två PWM-signalerna representeras då istället av två likspänningar med var sin amplitud. Därefter förstärks spänningarna för att på så vis kunna jämföras mot en extern referensspänning.

Referensspänningen väljs till en nivå mellan de två spänningsnivåerna för att på så sätt kunna avgöra om den lågpassfiltrerade PWM-signalen är högre eller lägre än referensspänningen. Om spänningen hos signalen är högre än referensspänningen sätts spänningen till transistorn hög. Om spänningen hos signalen är lägre än referensspänningen sätts spänningen till transistorn till låg. Designparametrar gör att spänningen ifrån transistorn till multiplexern ger ett inverterat beteende, det vill säga då spänningen till transistorn sätts till hög fås låg signal till multiplexern och vice versa.

Signalen som sätts hos multiplexern är en signal kallad ”set” som avgör vilken av två ingångar som ska sättas till utgången. Setsignalen bestämmer detta för fyra par vilket gör att fyra utav åtta ingångar sätts till utgången. Med den här designen erhålls en krets som urskiljer två stycken PWM-signaler, baserat på omkopplarens läge på RF-sändaren, för att på så vis avgöra vilken av insignalerna till multiplexern som ska väljas.

För att erhålla önskad funktionalitet studerades multiplexerns insignal. Signalen från RF-mottagaren som skulle urskiljas studerades i oscilloskop, se Figur4.5. Periodtiden uppmättes till 20 ms (Figur 4.5a), pulstiden var 1 ms (Figur 4.5b) respektive 2 ms (Figur 4.5c) beroende på omkopplarens läge, och spänningsnivån var 3,3 V. En sammanställning av mätdata från oscilloskopsobservationerna kan ses i Tabell 4.2.



(a) Signal, översikt **(b)** 0 % utslag **(c)** 100 % utslag

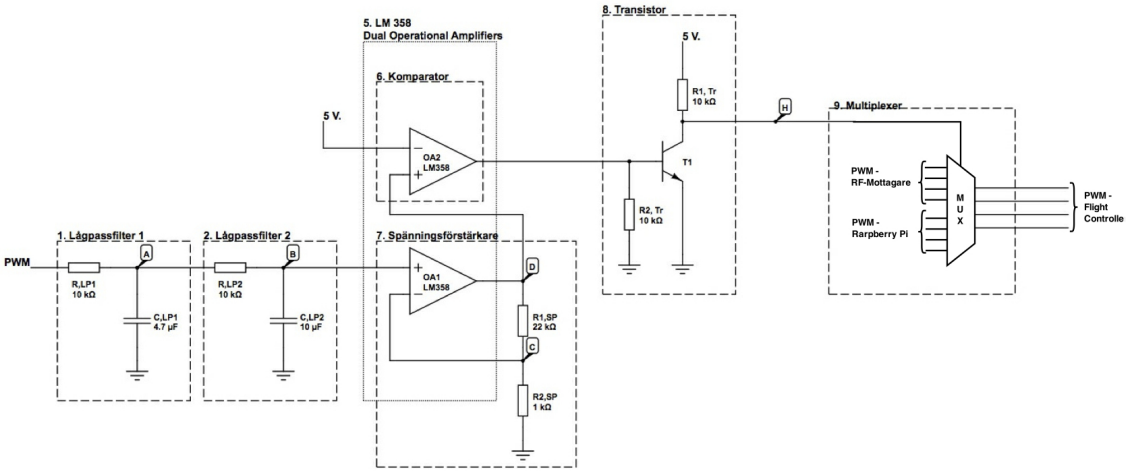
Figur 4.5: Signal från RF-mottagaren studerat i oscilloskop

Tabell 4.2: Mätdata från RF-mottagaren

Omkopplare (Läge)	Utslag [%]	Pulstid [ms]	Periodtid [ms]	Pulskvot [%]
1	100	2	20	10
2	0	1	20	5

Ett kopplingsschema för kretsen kan beskådas i Figur4.6. För att enklare kunna utföra tester på kretsen genererades likvärdiga signaler med hjälp av en Arduino

uno med tillhörande egenskrivet program. PWM-signalen lågpassfiltreras med två stycken kaskadkopplade första ordningens lågpassfilter som tillsammans bildar ett andra ordningens lågpassfilter.



Figur 4.6: Kopplingsschema för multiplexern

Spänningsnivåerna förstärktes 23 gånger enligt ekvation 4.1, med en operationsförstärkare (OP) kopplad som en icke inverterande förstärkarkoppling.

$$V_D = V_B \frac{R1_{SP}+R2_{SP}}{R2_{SP}} \tag{4.1}$$

Detta gav de två nya spänningsnivåerna 3 V respektive 6,5 V. Spänningarna jämfördes med en OP enligt ekvation 4.2. Då en OP har väldigt hög förstärkning kommer OP:n att bottna redan vid små differenser. Utspänningen blir då strax under OP:ns matningsspänning VCC. Eftersom OP:n endast matas med positiv spänning kommer den som lägst ge 0 V ut. Trots sin stora förstärkning krävs en större spänningskillnad än differensen mellan de endast lågpassfiltrerade signalen, därav spänningsförstärkningen.

$$V_{out} = A_{open\,loop} \cdot (V_+ - V_-) \tag{4.2}$$

Med spänningsförstäkningen minskade dessutom kretsens känslighet mot spänningsvariation. Då spänningsnivån ökar påverkas endast spänningen till gaten på transistorn och då maxspänningen för denna är 30 V finns god säkerhetsmarginal. Om spänningsnivån skulle sjunka ner till 8 V påverkas på nytt endast gatespänningen som återigen inte påverkar kretsen då transistorn leder fullt redan vid 1,8 V. Under 8 V sjunker den övre spänningsnivån, det vill säga den lågpassfiltrererade PWM-signalen som jämförs vid komparatorn, eftersom spänningsförstärkaren är bottnad. Då spänningen sjunker ytterligare och närmar sig referensspänningen finns det risk

att spänningsnivån alltid förblir lägre än referensspänningen och transistorn kommer då aldrig att leda.

Genom att förstärka spänningsnivåerna kunde en referensspänningsnivå på 5 V väljas vilket underlättade då denna redan fanns att tillgå i quadcoptern. Detta medförde att antalet komponenter kunde minimeras. För att garantera rätt spänning på set-signalen till multiplexern matades denna via en transistor. Transistorn bidrog till minskat spänningsberoende och säkerställde stabila spänningsnivåer. På grund av multiplexerns design valdes en inverterande funktion med transistorn.

Resistorstorlekarna är valda utifrån att kretsen inte ska dra någon betydande ström utan endast jämföra och förstärka spänningnivåer. Då kretsen inte förbrukar någon större effekt gjordes valet att använda ett så kallat ”pull-down motstånd” på transistorns gateingång för att säkerställa att ett flytande läge inte uppnås, det vill säga en spänningsnivå som varken är låg eller hög.

5

Driftsättning, testning och verifiering

Efter att quadcopterns ingående delar hade blivit individuellt implementerade och-testade kopplades de ihop för testning och verifiering av dess helhetsfunktion. Testergjordes för att verifiera att hårdvaran fungerade och att programmen kommunicerade och loggade önskad data. En enkel modellering och reglering utfördes för att undersöka quadcopterns potential till autonom styrning.

5.1 Mekatronisk testning

Under projektets tidiga stadie var det ett antal komponenter som inte presterade säkert och stabilt och behövde därför förnyas. Efter tester där olika produkter jämfördes blev gamla komponenter ersatta av nya införskaffade komponenter. När alla komponenter fanns att tillgå, det vill säga då de nya komponenterna levererats, genomfördes olika funktionstest av delar i styrsystemet. Test för att fastställa grundläggande funktioner så som att skicka vissa signaler samt påverkan på gyroskopet genomfördes innan allt kopplades upp med rotorblad och matades med batterispänning för ett flygtest.Under flygtestning upptäcktes ett problem där motorerna inte var fäst permanent utan kunde till viss del rotera kring fästet. Åtgärder för att göra dem mer stabila genomfördes och manuella flygtest repeterades för att säkerställa att förbättring hade uppnåtts. Under ytterligare flygningar uppnåddes stabilitet hos quadcoptern.

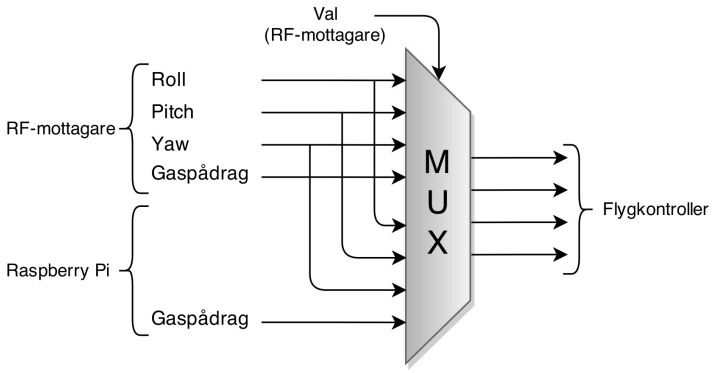
För att optimera flygkontrollerns respons och prestanda behövdes dess interna PID-regulator ställas in då systemet påverkas av placeringen hos quadcopterns komponenter. Efter informationshämtning på internet erhölls riktlinjer angående hur optimering av flygkontrollerns interna kaskadkopplade PID-regulator genomförs [37]. Optimeringen genomfördes med ett flygtest där de olika PID-parametrarna justerades för att erhålla de mest stabila resultaten av flygningen. Detta gjordes för att underlätta styrning manuellt och möjliggöra att flygkontrollern får quadcoptern att plana ut och hovra om inga signaler skickas till den. Framtagna parametrar åskådliggörs i Tabell 5.1. Dessa parametrar påverkar i sin tur accelerationsmöjligheterna för quadcoptern, gör den mer respektive mindre stabil samt påverkar manövreringen.

Tabell 5.1: Tabell över interna PID parametrar i flygkontrolleren

Parameter	P gain	P Limit	I gain	I limit
Roll	180	100	105	20
Yaw	180	20	20	10
Self level	100	20	-	-

5.2 Driftsättning och inledande funktionstest

När alla program, sensorer och hårdvara installerats och implementerats kopplades allt samman. Iakttagelser gjordes kontinuerligt för att säkerställa att tillfredsställd funktion uppnåddes. Efter driftsättningen var systemidentifikation av quadcopterns betende i höjd det första testet som genomfördes. För att testet skulle kunna utföras säkert manövrerades endast höjden av Raspberry Pi:n. Detta åstadkoms genom att koppla in signalerna från RF-mottagaren till multiplexern. Alla signaler utom höjd dubblerades och kopplades in på multiplexerns inportar för Raspberry Pi enligt Figur 5.1. Genom att endast koppla in höjdsignalen från Raspberry Pi:n till multiplexern kunde två lägen väljas. Antingen styrdes quadcoptern enbart från RF-sändaren eller så styrdes höjden av Raspberry Pi:n och resten av RF-sändaren. På så sätt kunde alltid styrning över quadcoptern återtas manuellt.



Figur 5.1: Inkoppling av multiplexer vid höjdtest

Raspberry Pi:n, multiplexern och flygkontrollern kräver 5 V matningsspänning och dessa försörjdes från en av motordrivarna. Vid detta laget var varken 3G-modemet eller GPS:en inkopplade till Raspberry Pi:n. Motordrivaren blev då väldigt varm och driftsättningen avbröts genast och utvärderades. Slutsatsen var att så länge motordrivaren inte blev överhettad skulle den inte ta skada men att åtgärder var tvungna att vidtas. För att komma runt problemet införskaffades en spänningsregulator. Strömförsörjningen kopplades sedan om så att alla komponenter som kräver 5 V matades från denna. När quadcoptern åter spänningssattes fungerade all hårdvara som planerat.

Under ett test av magnetometern roterades quadcoptern. Då uppstode de ett problem då spänningsmatningen försvann korta stunder ifrån flygkontrollern. Bristfälliga anslutningar antogs vara anledningen och efter nya lödningar på spänningsregulatorn försvann problemet. Spänningsförsörjningen delades senare upp i två separata delar där Raspberry Pi:n matas via spänningsregulatorn som matas från batteriet medan resten matas via BEC från motordrivaren för att höja säkerheten.

5.3 Testning och verifiering av plattform

I den slutgiltiga lösningen är ett flertal tester implementerade för att testa plattformens sensorer och delfunktoner. Det finns tester utformade för enstaka sensorer, grupper av sensorer samt tester av quadcopterns flygbarhet. Även testfall för PID-reglering i höjdled finns implementerade, se Appendix B.

Dessa test utfördes både när bara Raspberry Pi:n och sensorn var inblandade samt under flygning då sensorerna var fästa på quadcoptern. Plattformen har verifierats genom dels tester av de monterade sensorerna och dels genom testflygningar där alla system arbetat ihop.

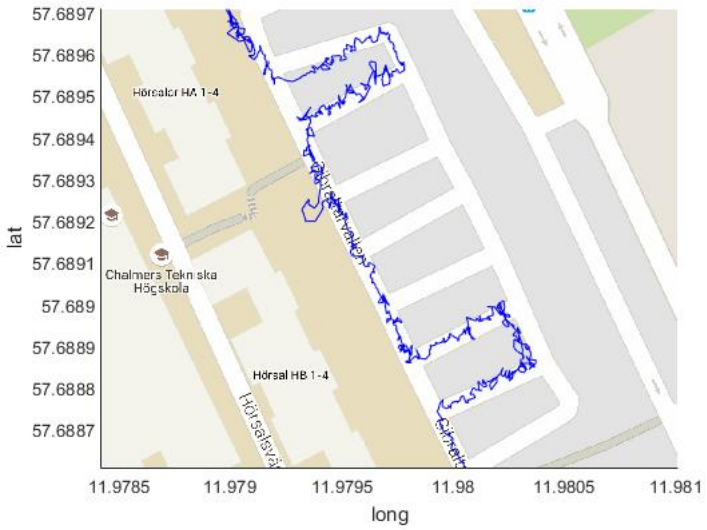
Då ett flygtest med sensorer skulle genomföras för första gången och allt var uppkopplat blev det kortslutning mellan ultraljudsensorn och en skruv på ramen vid kalibrering av magnetometern. Detta resulterade i att Raspberry Pi:n överhettades. För att säkerställa att detta inte skulle ske igen isolerades all komponenter med isoleringstejp och fästes ordentligt.

GPS

Inledande tester av GPS:en i Windows-miljö visade att enheten fungerar korrekt. I Figur 5.2 är en kort sträcka på en parkeringsplats loggad där den riktiga färden följde räta linjer i de former som kan antydas i figuren.

I ett försök att eliminera oscillationerna av positionsangivelser lågpassfiltrerades denna data, efter att avvikande värden ignorerats. Undersökningar av GPS:en på dator visade att denna metod kraftigt slätade ut positionsangivelserna, dock till priset av att positionsangivelsen fördröjs i den filtrerade datan, där den rapporterade positionen kan ligga ett antal sekunder bakom den verkliga. I Figur 5.3a är ett enkelt glidande medelvärdesfilter implementerat samt kraftigt avvikande mätpunkter ignorerade.

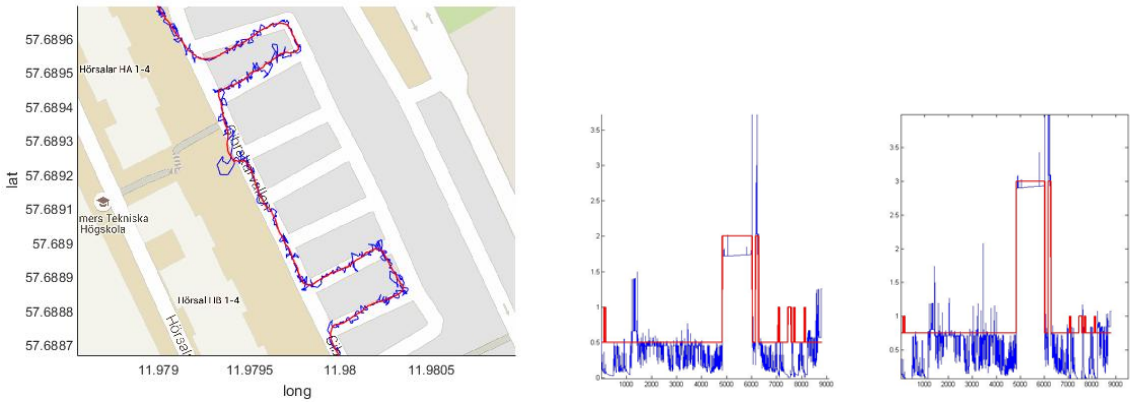
De avvikande punkter som slängts bort är baserade på RTKLIB:s avvikelserapportering som är del av utdatan och uppskattar standardavvikelse i longitud samt latitudsangivelser. Beroende på vilken kvalitétsflagga som RTKLIB rapporterade så filtrerades punkterna olika. I detta fall valdes värden bort som hade avvikelser i longitud på 0.5 och latitud på 0.75 standardavvikelser vid en FLOAT-fix, som är den näst bästa. Vid den bästa kvalitétsflaggan FIX så kastades punkter med mer än 1 standardavvikelse bort, dock så fanns inga sådana punkter i denna mätserie. Vid



Figur 5.2: Ofiltrerad GPS-data, 5Hz

den sämre SINGLE så kastas punkter över 2 respektive 3 standardavvikelser bort, då ett striktare krav hade inneburit att alla punkter skulle kastas bort. Som visas i Figur 5.3b är mätosäkerheten större i nord-sydlig riktning än öst-västlig riktning.

Alla värden över den röda grafen i Figur 5.3b förkastas i positionsangivelsen på Figur 5.3a. Notera att Figur 5.2 och Figur 5.3a endast täcker in en del av mätdata som 5.3b visar.



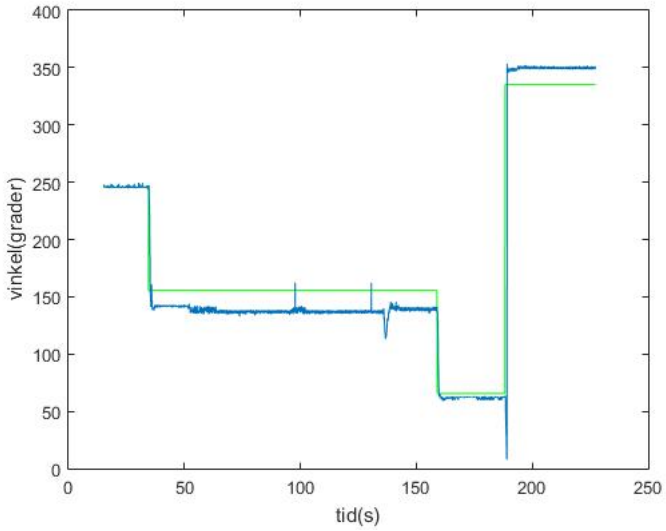
(a) Filtrerad och behandlad GPS-data, 5Hz

(b) Standardavvikelse i nord-sydlig till höger och öst-västlig till vänster

Figur 5.3: Filtrerad GPS-data, 5 Hz

Magnetometer

Magnetometerns funktion verifierades genom att undersöka loggar och utdata i realtid. Magnetometern ställdes in på 0° och roterades sedan med etapper om 90°. Det kunde då verifieras att rapporterade gradtal stämt överens med den roterade vinkeln. Precisionen sjunker då magnetometern roteras ur sitt horisontalläge, eftersom endast de horisontella axlarna används i beräkning av bäring. Som ses i Figur 5.4 så följer mätdatan den förväntade datan väl förutom vid rotation på 90° och 270° eftersom underlaget som användes lutade 3°.



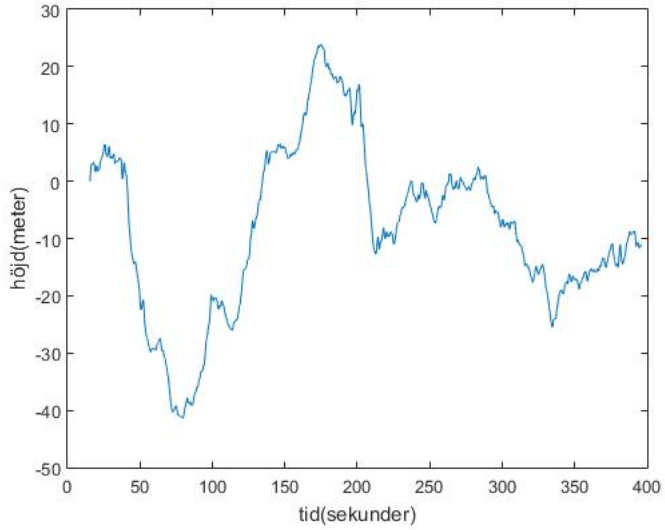
Figur 5.4: Magnetometerdata vid rotering om 90° på ett 3° lutande underlag

Barometer

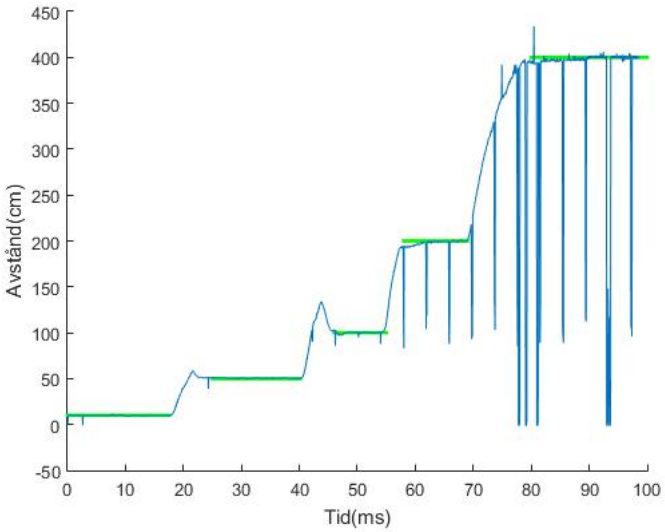
Barometerns funktion verifierades genom att undersöka loggar och följa den i realtid. Barometerns absoluta höjd kan inte bestämmas utan att veta det aktuella trycket vid havsnivån, men genom att utgå ifrån ett ungefärligt värde så kan barometerns höjd relativt marken beräknas. Barometerns rapporterade lufttryck förändras trots att höjden faktiskt inte förändras, och kan typiskt förändras upp till 20 meter, se Figur 5.5.

Ultraljudssensor

Ultraljudssensorns funktion har verifierats genom loggning och undersökningar i realtid. I Figur 5.6 hålls ett föremål på 10, 50, 100, 200 och 400 centimeters avstånd från ultraljudssensorn. De gröna linjerna motsvarar referensavståndet uppmätt med tumstock.



Figur 5.5: Mätning av höjd med barometer på konstant en höjd.

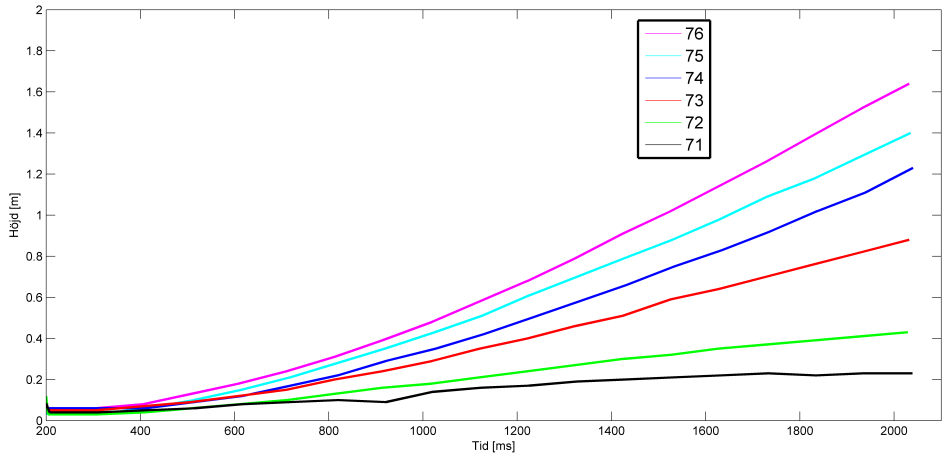


Figur 5.6: Ultraljudssensorns avståndsrapportering vid 10, 50, 100, 200 och 400 cm

Systemidentifikation

Programstrukturen har verifierats genom att undersöka signaler från alla sensorer i realtid, för att se att kommunikation mellan de olika processerna fungerar. För att modellera quadcopterns rörelse i höjdled utfördes ett test där ultraljudssensorn loggade höjd och tid i ett testprogram. Programmet skrevs så att quadcoptern utgick från ett konstant gaspådrag motsvarande hovring från Raspberry Pi:n. Efter ett par sekunder ökade gaspådraget med ett förbestämt steg och mätdata från ultraljudssensorn samlades in. Testet upprepades med flera olika storlekar på gaspådraget. Utifrån mätdatan kunde höjdens beroende av tiden för ett visst gaspådrag visualiseras, se Figur 5.7.

Höjden mättes med tio mätpunkter per sekund. Resultaten visas i Figur 5.10 med olika steg och dess höjd beroende på tiden. Effekten motorerna levererar är beroende av batterispänningen och testen genomfördes under hela batteritiden. Figur 5.10 representerar då systemet med ett medelvärde av batterispänningen.



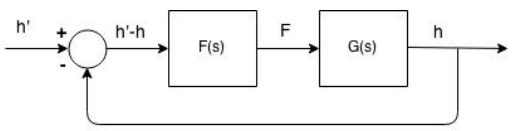
Figur 5.7: Höjd efter gaspådrag mätt i procent av max, kraft innan steg på 69

Med dessa värden så kan en verklighetsförankrad modell av quadcoptern tas fram. För att testa en möjlig implementering av en regulator för höjdreglering skrevs det kod som sedan kunde användas. Koden var en diskretisering av en PID, vilket använde uppdateringshastigheten för beräkning av integralverkan och skillnaden mellan två mätpunkter för beräkning av derivataverkan.

5.4 Modellering och reglerdesign i höjdled

För att quadcoptern ska vara medveten om sin position krävs reglering i tre led; horisontell position, höjd samt bäring. En modell för systemet bör innehålla styrsignaler (yaw, roll, pitch, gaspådrag) från Raspberry Pi:n till flygkontrollern.

Systemet i höjdled kan modelleras på flera sätt, varav ett utgår ifrån Newtons andra lag: $F = ma = m\ddot{h} \overset{L}{\leftrightarrow} F(s) = ms^2H(s) \rightarrow G(s) = \frac{1}{ms^2}$ vilket kan användas i ett återkopplat reglersystem enligt Figur 5.8

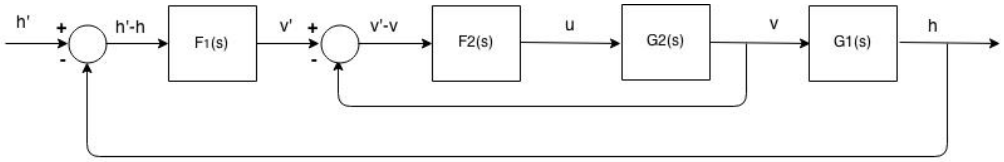


Figur 5.8: PID regulator

Det gäller då att reglera lyftkraften som utvecklas i motorerna med gaspådrag till flygkontrollern. Flygkontrollern innehåller dock intern reglering och implementerar inte nödvändigtvis ett linjärt samband mellan gaspådrag och lyftkraft.

Efter ett antal tester framgick det att flygkontrollerns interna reglering snarare kunde liknas vid ett samband mellan gaspådrag och hastighet, det vill säga att flygkontrollern reglerar för en konstant hastighet vid ett givet gaspådrag.

En reglering kan implementeras i kaskad med flygkontrollerns interna reglering enligt Figur 5.9 där F2 och G2 representerar flygkontrollerns interna reglering. F1 reglerar en höjdskillnad till en önskad hastighet, och G2 motsvarar den inverkan en hastighet i höjdled har på systemet.

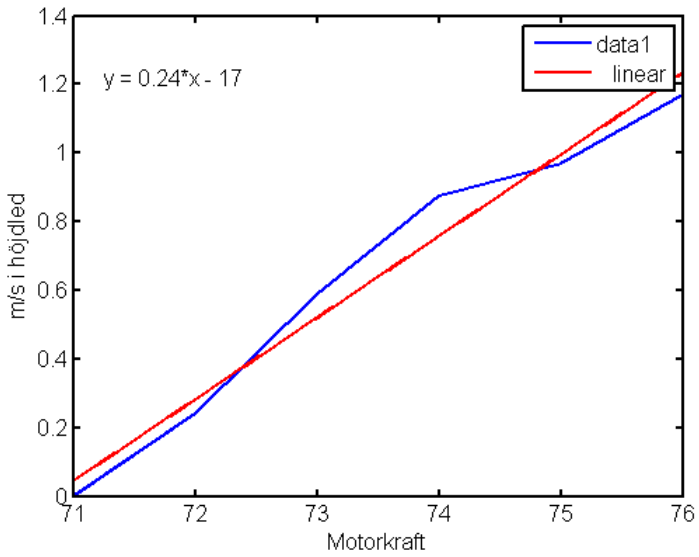


Figur 5.9: Kaskadreglerat system där flygkontrollerns interna reglerloop är uppskattad

Denna metod för höjdreglering var den som slutligen valdes, eftersom det i test framgick att ett visst gaspådrag till flygkontrollern snarare gav utlopp till en konstant hastighet än en accelererande rörelse, se Figur 5.7. I Figur 5.10 är olika gaspådrag ritade mot den hastighet som de motsvarade i Figur 5.7 och en linjär regression är gjord i MATLAB. Detta resulterade i ett förstagradspolynom $v = 0.24x - 17$ där v är hastighet i höjdled och x ett gaspådrag i procent av maxgas. Genom att utgå ifrån att quadcoptern hovrar, kan konstanttermen tas bort (71% tolkas som origo) och den nya funktionen blir $v = 0.24x$. Med detta samband kan vi beskriva systemet genom en laplacetransform $v = 0.24x \leftrightarrow \dot{h} = 0.24x \overset{L}{\leftrightarrow} sH(s) = 0.24X(s) \leftrightarrow$

$$G1(s) = \frac{0.24}{s} \tag{5.1}$$

Modellen testades genom att ett konstant gaspådrag lades på för att kompensera för gravitation och med en implementerad regulator kunde quadcoptern hovra på en låg höjd. Resultatet blev ett överslag som snabbt kompenserades för och referenshöjden hittades. Det var bara små oscillationer när quadcoptern först nått referenshöjden.



Figur 5.10: Olika gaspådrag plottad mot resulterande hastighet.

6

Diskussion och förbättringsmöjligheter

Under arbetets gång föddes insikten att det i vissa fall fanns både lättare och bättre sätt att ta sig an och lösa olika typer av problem. Vidare väcktes tankar kring möjliga utvecklingar av ingående delar. Det här kapitlet behandlar diskussion av resultat samt redogör för såväl delar med förbättringpotential som vidareutvecklingar av arbetet.

6.1 Hårdvara

Efter att initialkonstruktionen av den mekatroniska delen var färdigställd och hade testats fungerade den som planerat. När hovringsfunktionen senare testades upptäcktes dock ett problem. Batterispänningen varierar med användning och resulterar i sjunkande motorkraft för samma gaspådrag. När ett tre-cells litiumbatteri är fulladdat ligger spänningen på ca 12,5 V, och vid 11,1 V börjar spänningen att falla kraftigt. Detta innebär att det krävs olika gaspådrag för att uppnå hovring vilket försvårar regleringen. För att hålla spänningen konstant är en möjlig lösning att mata quadcoptern via en spänningsregulator och sänka spänningen till 11,1 V då den egentligen är högre. Detta begränsar motorkraften, men gör systemet mer stabilt. En annan lösning är att en framtida implementerad regulator tar hänsyn till sjunkande spänning vid drift och kompenserar för detta.

6.1.1 Ultraljudssensor

I Figur 5.6 syns vissa dalar i mätpunkterna. En del av dessa kan bero på att ett längre avstånd än 400 cm uppmätts, varför den enligt från drivrutinen BMP180.c rapporterar -1. Att vissa dalar inte går ända ned till -1 beror på att mätvärdena har filtrerats genom ett filter, av typen glidande medelvärde. En annan anledning till varför avståndet rapporterats som -1 kan vara att huvudprogrammet försöker läsa av datan innan sensorprocessen hunnit göra datan tillgänglig, vilket sannolikt skulle kunna förklara de periodiska tendenser som syns i figuren.

Ultraljudssensorn har maximalt en mätosäkerhet på 3 millimeter och kan mäta avstånd upp till 5 meter. Detta gäller mot platta och hårda ytor. Vid tester mot gräs, och vid längre avstånd eller lutade plan kan problem med avståndmätningen uppstå. Det är även möjligt att vind kan påverka mätningarna. En laseravstånds-

mätare hade troligen fått en bättre noggrannhet i flertalet scenarion men då till ett högre inköpspris.

6.1.2 Barometer

De kraftigt drivande värden som kan observeras i Figur 5.5 beror antagligen på ett fel i implementering av mjukvara. Då barometerns mätosäkerhet är av sådan magnitud (tiotals meter) samt att barometern testats tidigare på en annan mikrokontroller med goda resultat, är en felaktig implementation av någon del i bmp180.c de troligaste felet. Vid noggrannare undersökningar av temperaturmätningen visade det sig att temperaturen inte uppdaterade sig på ett korrekt sätt. Baserat på detta är den troligaste orsaken till barometerns kraftiga mätosäkerhet orsakad av felaktig temperaturmätning.

Barometern påverkas även av yttre störningar och dess mätvärden kan skilja många meter utan att barometern flyttas. Störningar så som temperatur och kastvindar har simulerats och resulterat i att beräknad höjd har ett mätfel på upp till 10 meter. Störningarna kunde eventuellt dämpats om sensorn täckts av skumplast likt det som återfinns kring en mikrofon.

6.1.3 Magnetometer

Magnetometern som användes i projektet är en relativt billig modul men har en acceptabel mätosäkerhet på 1 grad [45]. Magnetometern fungerar bra för att beräkna en kompassriktning då den ligger plant. Detta fungerar bra vid stationär testning men under flygning kommer magnetometern sällan vara plan och ger då felaktiga mätvärden vilket gör den nuvarande implementationen oanvändbar för praktiska tillämpningar. För att komma till rätta med problemet finns två möjliga lösningar. En lösning är att med en mekanisk rigg se till att magnetometern alltid är placerad plant. Detta är fullt möjligt men kan vara svårt att implementera samt uppta mycket plats. Den andra metoden är att komplettera magnetometern med en accelerometer som mäter quadcopterns lutning. Med hjälp av quadcopterns roll och pitch kan en exakt kompassvinkel räknas ut med följande ekvationer där Θ är roll och φ är pitch. [46]

$$Xh = X * cos(\varphi) + Y * sin(\Theta) * sin(\varphi) - Z * cos(\Theta) * sin(\varphi) \tag{6.1}$$

$$Yh = Y * cos(\Theta) + Z * sin(\Theta) \tag{6.2}$$

Xh och Yh från 6.1 och 6.2 kan sedan använs i den nuvarande implementationen av kompassriktning istället för X och Y. [46]

6.1.4 GPS

GPS:en har en mätosäkerhet på upp till tio meter men varierar beroende på omgivning. Detta innebär begränsningar inom användningsområden och kräver att flygning sker på öppna ytor. Detta för att undvika risken att flyga in i föremål då

mätosäkerheten är stor.

För att förbättra precisionen gällande quadcopterns position finns olika åtgärder att vidta. Ett alternativ är att införskaffa en antenn med högre förstärkning. Trots ett sådant inköp är det inte säkerställt hur precis datan blir. En annan åtgärd är att införskaffa en ny GPS-modul. Det som krävs av denna GPS-modul är att den förutom traditionell GPS även stödjer GLONASS, vilket NEO-6t inte gör. Detta skulle innebära att signaler kan tas emot från fler och mer synliga satelliter vilket ökar möjligheterna till förbättrad precision. En kombination av dessa två åtgärder ger troligtvis störst förbättring.

6.1.5 Kommunikation

Det har implementerats två sätt att kommunicera med Raspberry Pi:n. Dessa är mobilt 3G-modem [47] och WiFi-dongel [48]. WiFi-dongeln används för att koppla Raspberry Pi:n till en dator med ssh via en telefon som delar ett nätverk. Telefonen kan ersättas med en router för bättre täckning. Med ett 3G-modem kan uppkoppling mot nätet också upprättas, men då måste andra system än ssh implementeras för att skicka information.

6.2 Mjukvara

I början av projektet valdes C som programmeringspråk av två anledningar:

- RTKLIB är skrivet i C och skulle därför kunna inkluderas i huvudprogrammet för enkel kommunikation [17]
- C är ett traditionellt språk för lågnivåprogramering och är resurssnålt

Under projektets gång har detta val visat sig vara icke optimalt. Då systemet för sensorhantering består av parallella processer skulle de olika processerna kunna vara skrivna i olika språk. Även argument att RTKLIB skulle inkluderas i huvudprogrammet skiljer från den faktiska implementationen då RTKLIB innehåller funktioner för att kommunicera med andra program. Om ett högnivåspråk hade valts skulle antagligen tiden för att producera koden kortats ner. Två alternativa språk skulle antagligen varit lämpligare val för hela programstrukturen eller som en del av den. Ett språk som kunde underlättat vore python eftersom det finns färdiga bibliotek för projektets sensorer vilket skulle sparat utvecklingstid. Ett annat val som skulle sparat tid är att skriva kod i Java, då gruppens medlemmar har mer vana av programmering i detta jämfört med C. Ett högnivåspråk skulle även enklare kunna hantera program med flera trådar vilket skulle eliminera behovet av att köra flera program samtidigt.

Själva implementationen med parallella processer i C hade också kunnat förbättras. De pipes som används för att skicka data mellan processer kommer med ett antal begränsningar som måste åtgärdas i koden. Ifall huvudprogrammet skickar begäran

om data snabbare än vad drivrutinen kan läsa data riskerar bufferten till pipen att fyllas och låsa programmet. Det finns också en risk att en sensorprocess inte hinner uppfylla en databegäran innan huvudprogrammet läser av pipen, i vilket fall detta värde kommer att ligga kvar på pipen tills den läses nästa gång. Huvudprogrammet riskerar då att få gamla värden, dock maximalt en mätning gamla.

6.3 Design och säkerhet

Quadcopterns design är oflexibel då de flesta komponenter är permanent monterade. Dess elektriska utrustning är oskyddad mot fukt vilket gör den sårbar och begränsar användningen. Dessutom kan quadcopterns tyngdpunkt variera något på grund utav batteriets möjlighet att flytta sig. Komponenternas täta placering är fördelaktig då rörelse i de olika rotationsriktningarna underlättas.

En möjlig utveckling gällande designen är att arbeta fram ett komponenthus där alla komponenter kan fästas i hakar och därmed även plockas loss vid behov. Det behöver vara möjligt att placera komponenterna kompakt, utan risk för kortslutning. Ett tätt, heltäckande skal för komponenter och ram skulle vara fördelaktigt, inte bara ur ett estetiskt perspektiv utan framför allt ur säkerhetssynpunkt. Skalet skulle skydda alla komponenter och sladdar från rotorbladen, minska risken för skador vid eventuella krascher eller krockar samt skydda mot dåliga väderförhållanden så som regn och snö. Vidare skulle det täta skalet möjliggöra landning i exempelvis fuktigt eller blött gräs då komponenterna inte riskerar att komma i kontakt med vätan. Skalet skulle även skydda användaren och omgivningen vid eventuella batteriproblem så som brand eller explosion.

En ytterligare utveckling på quadcoptern är att designa ringar att fästa på varje arm. Ringarna sitter runt respektive rotorblad och förhindrar att någon eller något i quadcopterns omgivning skadas vid flygning. Vidare skyddar ringarna även rotorbladen vid flygning i områden där quadcoptern riskerar flyga emot fasta föremål.

Olika säkerhetsaspekter för såväl person som maskin har alltid tagits i beaktning då beslut fattats. Fokuset på säkerheten har öppnat upp för möjligheter till att utföra diverse användbara tester på ett kontrollerat sätt. Möjligheten att överta styrningen via RF-sändaren har medfört att verkliga tester kunnat utföras. Med hjälp av noggrant skrivna testprogram och väl övervakad data och quadcoptern har olika delfunktioner testats och vid antydningar till problem eller komplikationer har styrningen direkt övertagits av en operatör. Operatören har i dessa situationer säkert och kontrollerat kunnat styra upp quadcoptern och därefter landa vid behov.

En annan möjlig utveckling av säkerheten är att voltmätaren som mäter batterinivån kopplas till Raspberry Pi:n. Med den förutsättningen kan ett program skrivas på Raspberry Pi:n för att landa quadcoptern säkert utan att insats från operatör krävs. Vidare skulle Raspberry Pi:n även kunna beräkna om flygrutten kan avslutas beroende på hur långt det är kvar, om quadcoptern kan återvända till startpunkten och landa, eller om landning krävs omedelbart.

6.4 Måluppfyllnad

Flygtester visade att quadcopterns manuella styrning fungerar. Detta innebär att implementeringen av flygkontrollern och motordrivarna lyckades. Som följd av detta uppnåddes alltså målet att implementera nödvändiga komponenter för att kunna manövrera quadcoptern manuellt.

Mätdata från de olika sensorerna kan hämtas med de olika egenskrivna drivrutinerna med varierande resultat som på olika sätt kan förbättras. Data från GPS:en kan inhämtas men kräver vidare bearbetning i huvudprogrammet för att möjliggöra positionsbestämning. Detta innebär att alla sensorer kan kommunicera med mjukvaran vilket medför att målet gällande kommunikation uppfylldes. Som följd av att ytterligare bearbetning av GPS-data krävs uppfylldes dock inte målet gällande hantering av data.

Vidare visade tester av höjdreglering att autonom flygning i höjdled inom ultraljudssensorns arbetsområde fungerar. Under testet användes säkerhetssystemet (multiplexern) för att ändra ingångskälla till flygkontrollern och överta manuell styrning. Detta bekräftade att såväl säkerhetssystemet som funktionen mellan hård- och mjukvara fungerar för autonom styrning i höjdled upp till fyra meter.

Höjdledstestet visade även att quadcoptern autonomt kan bibehålla sitt tillstånd luften. Den data som krävs för att kunna möjliggöra autonom styrning i andra led finns att tillgå men kräver, som tidigare nämnts, vidare bearbetning. Detta innebär att autonom styrning är möjlig med de komponenter som plattformen i slutskedet innehöll. Följaktligen uppfylldes alltså målet att implementera den hårdvara som krävs för att möjliggöra autonom styrning.

Sammanfattningsvis uppnåddes alltså de flesta delmålen och endast förhållandevis lite arbete med mjukvaran kvarstår för att uppnå resterande mål. En fullständig måluppfyllnad skulle dock inte garantera god precision men det finns goda möjligheter att förbättra denna vid arbete med olika förbättringsförlag.

7
Slutsats

Utvecklingen av en plattform för autonom quadcopter mynnade ut i en fungerande produkt som kan samla in och bearbeta mätdata för att utifrån denna justera tillfört gaspådrag till motorerna. Hårdvarumässig implementering av GPS och sensorer möjliggör inhämtning av mätdata gällande quadcopterns position och höjd. Program upprätthöll kommunikation och bearbetning av datan. Utifrån tester användes datan till att skapa en modell för quadcopterns rörelse i höjdled. Genom att designa ett reglersystem för respektive rörelseriktning för quadcoptern är det möjligt att förverkliga autonom styrning.

Funktionstest av quadcopterns implementerade höjdreglering lyckades. Slutsatsen från testet är att quadcoptern har hårdvaru- och mjukvarumässiga förutsättningar att styras autonomt.

Resultaten från tester visade att plattformen var tillräckligt beräkningskraftig för autonom styrning. Ultraljudssensorn uppfyllde förväntningarna gällande noggrannhet och precision. Huruvida olika underlag påverkar dess mätosäkerhet är ännu inte klarlagt. Barometern har en mätosäkerhet på flertalet meter men har förbättringspotential. Nätverkstjänsten för RTK fungerar men GPS:en saknar stöd för GLONASS vilket försämrar dess precision. Dess mätosäkerhet kan förbättras med antingen en antenn med högre förstärkning eller med en ny GPS-mottagare.

Erforderliga mätvärden tillhandahålls av programmen som skrivits i form av olika processer som körs parallellt. Önskad mätdata kan via ett kommandotolksgränssnitt begäras från ett överordnat program. Detta förses med mätdata från olika program vilka individuellt kommunicerar och läser av sensorerna. Slutsatsen är att denna hierarki underlättar för fortsatt påbyggnad. Programmen är skrivna i C vilket i efterhand visade sig vara ett onödigt lågt val av programmeringsspråknivå. Samma mjukvarumässiga funktion kunde uppnåtts med andra typer av språk med ett större utbud av färdigskrivna funktioner för olika komponenter.

Källhänvisning

[1] www.oddcopter.com/flight-controllers/
Besökt: 18/5 20:49

[2] Krishnan Ramu (2009), Permanent Magnet Synchronous and Brushless DC Motors, CRC Press

[3] http://www.stefanv.com/electronics/qf200105.html
Besökt: 21/3 13:00

[4] http://www.stefanv.com/electronics/escprimer.html
Besökt: 21/3 13:00

[5] http://sv.wikipedia.org/wiki/Pulsbreddsmodulering
Besökt: 21/3 13:00

[6] http://www.adafruit.com/pdfs/raspberrypi2modelb.pdf
Besökt: 19/5 18:50

[7] http://www.nxp.com/documents/user_manual/UM10204.pdf
Besökt: 19/5 17:46

[8] Frisk, D. (2014) A Chalmers University of Technology Master's thesis template for LaTeX. Unpublished.

[9] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/
Besökt: 18/2 11:23

[10] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Fragor-och-svar/
Besökt: 18/2 11:23

[11] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/Swepos/
Besökt: 18/2 11:23

[12] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Metoder-for-GNSS-matning/RTK/
Besökt: 18/2 11:23

[13] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Metoder-for-GNSS-matning/Natverks-RTK/
Besökt: 18/2 11:23

[14] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Metoder-for-GNSS-matning/DGPS/
Besökt: 18/2 11:23

[15] http://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/Metoder-for-GNSS-matning/Felkallor-vid-GNSS-matning/
Besökt: 18/5 21:53

[16] http://wiki.openstreetmap.org/wiki/RTKLIB
Besökt: 18/5 20:23

[17] http://www.rtklib.com/
Besökt: 18/5 20:23

[18] http://gnss.co/?p=52
Besökt: 18/5 20:23

[19] http://www.raspberrypi.org/products/raspberry-pi-2-model-b/
Besökt: 22/3 12:30

[20] Rådfrågning av Mikael Tulldahl på Chalmers Robot Förening, 28/4 13:30

[21] https://github.com/adafruit/Adafruit_Python_BMP
Besökt: 5/5 20:30

[22] http://www.hobbyking.com/hobbyking/store/__39708__Afro_ESC_30Amp_Multi_rotor_Motor_Speed_Controller_SimonK_Firmware_.html
Besökt: 23/3 12:30

[23] http://www.siongboon.com/projects/2013-07-08_raspberry_pi/index.html
Besökt: 22/3 12:30

[24] http://www.micropik.com/PDF/HCSR04.pdf
Besökt: 19/5 08:40

[25] http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications
Besökt: 19/5 08:40

[26] http://www.ne.se.proxy.lib.chalmers.se/uppslagsverk/encyklopedi/lång/filter
Besökt: 25/3 19:30

[27] http://www.ti.com/lit/ds/symlink/lm358.pdf
Besökt: 31/3 18:30

[28] http://www.allaboutcircuits.com/vol_3/chpt_8/1.html
Besökt: 31/3 18:30

[29] http://www.ti.com/lit/an/sboa092a/sboa092a.pdf
Besökt: 31/3 18:30

[30] http://www.ne.se.proxy.lib.chalmers.se/uppslagsverk/encyklopedi/lång/operationsförstärkare
Besökt: 31/3 18:30

[31] http://www.kjell.com/sortiment/el/elektronik/mikrokontroller/arduino/avstandssensor-p87891#ProductDetailedInformation
Besökt: 2/5 14:20

[32] http://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf
Besökt: 2/5 14:30

[33] http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/ljudhastighet
Besökt: 2/5 14:40

[34] http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/lufttryck
Besökt: 2/5 15:00

[35] http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/magnetometer
Besökt 2/5 15:30

[36] https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/GPS-och-geodetisk-matning/GPS-och-satellitpositionering/GPS-och-andra-GNSS/Glonass/
Besökt: 27/4 17:30

[37] http://oddcopter.com/2013/01/06/easy-diy-quadcopter-build-part-4-tuning/
Besökt: 08/05 09:00

[38] http://ae-bst.resource.bosch.com/media/products/dokumente/bmp180/BST-BMP180-DS000-12 1.pdf
Besökt: 18/5 11:15

[39] https://github.com/richardghirst/PiBits/tree/master/ServoBlaster
Besökt: 18/5 15:34

[40] http://wiringpi.com/
Besökt: 18/5 16:12

[41] https://github.com/richardghirst/PiBits/tree/master/ServoBlaster
Besökt: 19/5 22:29

[42] http://www.hyperrealm.com/libconfig/
Besökt: 18/5 16:14

[43] http://www.advancedlinuxprogramming.com/alp-folder/alp-ch05-ipc.pdf
Besökt: 15/5 11:43

[44] https://www.m.nu/pdf/HC-SR04.pdf
Besökt: 18/5 19:10

[45] http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf
Besökt: 18/5 20:09

[46] https://www.parallax.com/sites/default/files/downloads/29133-Compass-Module-Application-Note.pdf
Besökt: 18/5 19:44

[47] http://consumer.huawei.com/en/mobile-broadband/dongles/features/e3372.htm
Besökt: 19/5 16:29

[48] http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/global/wireless_adapters_n150/ew-7811un
Besökt: 19/5 16:33

[49] https://github.com/antjohan/Quad
Besökt: 19/5 09:47

[50] http://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/glidande-medelv%C3%A4rde
Besökt: 18/5 22:16

[51] http://en.wikipedia.org/wiki/Six_degrees_of_freedom/media/File:6DOF_en.jpg

[52] http://publications.lib.chalmers.se/records/fulltext/200499/200499.pdf
Besökt: 19/5 22:21

[53] http://www.gnu.org/software/libc/manual/html_node/Pipes-and-FIFOs.htmlPipes-and-FIFOs
Besökt: 19/5 22:37

[54] http://docs.emlid.com/RTKLIB/rtklib-rover-setup/
Besökt: 29/4 2015

[55] http://raspberry.arctics.se/2013/03/30/ansluta-ett-3g-usb-modem-till-raspberry-pi/
Besökt: 21/3

[56] http://www.gp.se/motor/1.2725809-polisens-jakt-med-dronare-stoter-pa-patrull
Besökt: 29/5

[57] http://www.amazon.com/b?node=8037720011
Besökt: 29/5

A

Appendix - Uppstartguide

En del komponenter fanns att tillgå inom gruppen och kommer därför inte att följa med efter avslutat projekt. Appendix 1 innefattar en förklaring till de komponenter som kommande projektgrupp behöver införskaffa för att ta vid där det här projektet slutade. Dessutom ges en detaljerad beskrivning hur de olika komponenterna är inkopplade.

Nödvändiga komponenter

RF-sändare: En RF-sändare som kommunicerar på 2.4 GHz med minst 6 kanaler.

Internetdongel: För att upprätta kommunikation över internet krävs någon form av internetdongel. Till detta kan ett 3G-modem införskaffas då mjukvaran för detta är installerat. Alternativt kan en WiFi-dongel användas då quadcoptern befinner sig inom räckhåll från routern.

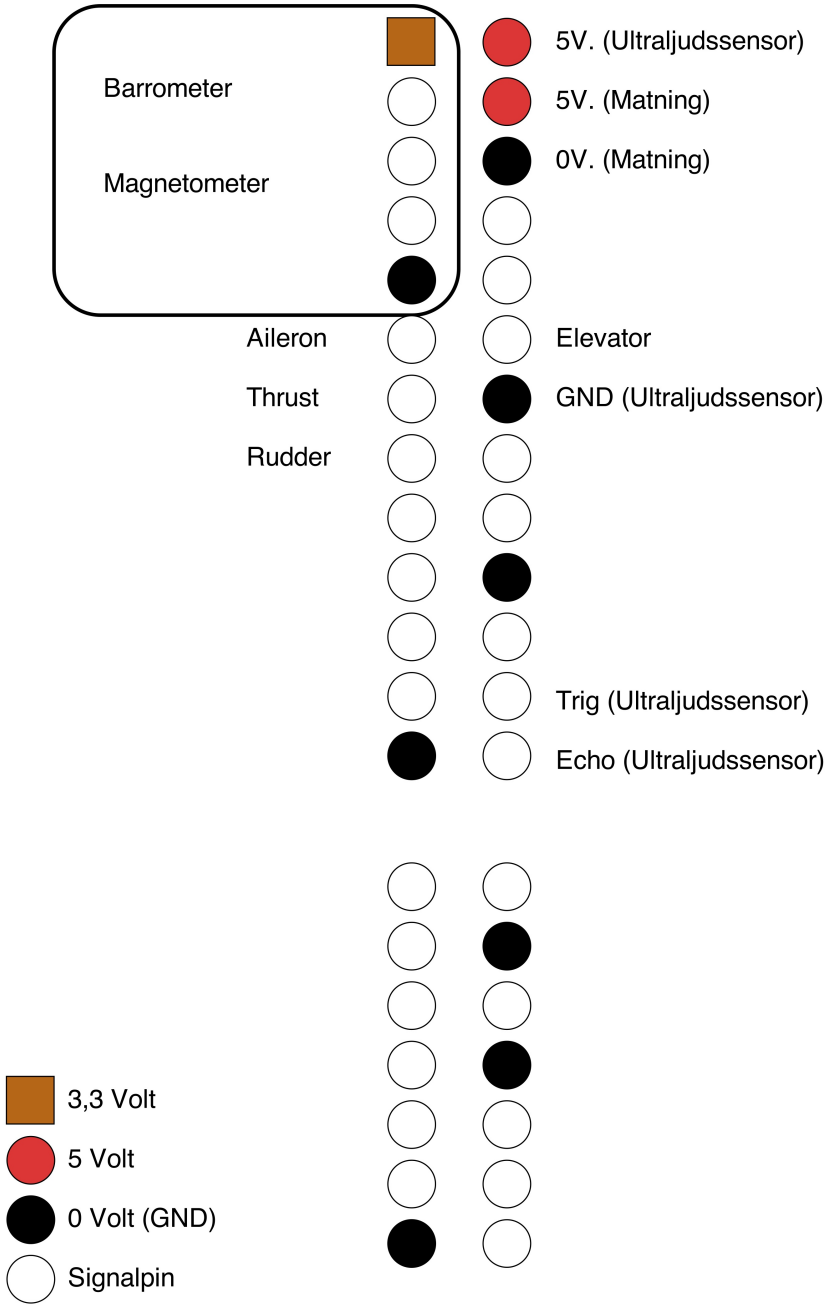
Voltmätare: För att veta när spänningen på batteriet som matar systemet sjunker är quadcoptern fördsedd med en voltmätare. Denna ger ifrån sig ett högt pip då spänningsnivån sjunker under en viss nivå vilket förhindrar oväntad minskning av spänningsnivån.

Inkopplingsbeskrivning

Raspberry Pi: Figur A.1 visar anslutningarna för Raspberry Pi och hur hur de olika komponenterna ska kopplas in. Vid anslutning får den fyrkantiga lödningen tas som referens. Till Raspberry Pi:ns USB ingångar kopplas GPS och lämplig internetdongel in.

Multiplexer: Inkoppling av signaler från RF-mottagaren och Raspberry Pi:n till multiplexern med dess tillhörande komponenter åskådliggörs i Figur A.2. De två ingångskällorna RF-mottagaren och Raspberry Pi:n kopplas in på A respektive B-ingångarna på kortet. Utgången, Y-utgångarna, kopplas till flygkontrollern. Ingångarna nedanför kortets utgångar är spänningsmatning, 5 V och batterispänning, signal ifrån RF-mottagaren och GND.

Flygkontroller: Signaler ifrån multiplexern kopplas till de vänstra ingångarna, ”Aileron” till ”Rudder”, på flygkontrollern enligt Figur A.3. De fyra motorerna på quadcoptern kopplas in på de högra ingångarna ”Motor 1” till ”Motor 4”. Ingången Äutolevel"kopplas till RF-mottagaren och erbjuder funktionaliteten autolevel när tillhörande omkopplare på RF-sändaren.

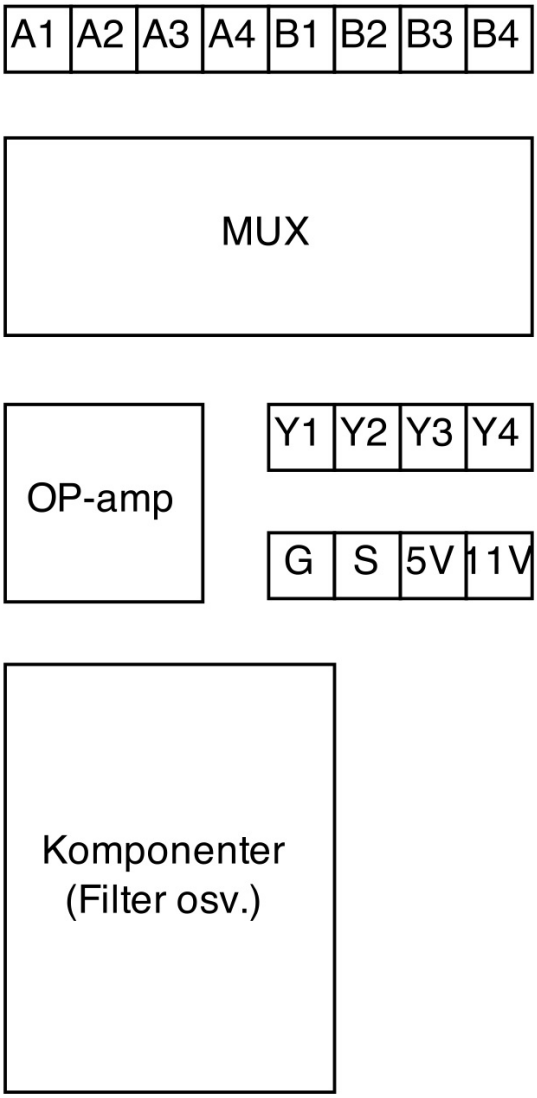


Figur A.1: Inkoppling av komponenter till Raspberry Pi 2.

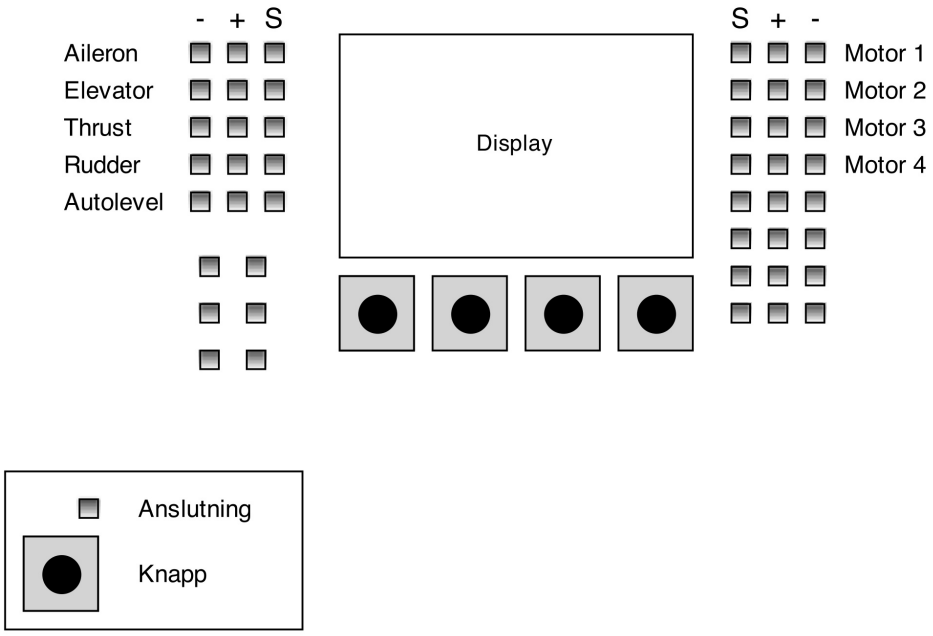
När IP-adressen för Raspberry Pi:n är känd kan åtkomst till denna ske via så kal-

lad ssh via terminalen(komandotolken) från en dator som är uppkopplad på samma nätverk. Efter att ha skrivit in användarnamn och lösenord, ”pi” respektive ”quad”, ges möjligheten att ändra, lägga till, ta bort och köra program.

För internetuppkopplingen till Raspberry Pi:n rekommenderas en WiFi-dongel. På Raspberry Pi:n ändras lämpligen en konfuigrationsfil för nätverket med kommandot ”sudo nano /etc/network/interfaces”. I denna ändras val av aktivt nät och eventuellt lösenord för nätverket.



Figur A.2: Inkoppling av MUX.



Figur A.3: Inkoppling av flykontroller.

B

Appendix - Programdokumentation

I projektet har kodning varit centralt för att skapa en fungerande plattform. Denna kod finns att tillgå [49] och är öppen för alla.

BMP180

Funktion: **void main();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Huvudfunktion, Initialiserar I^2C, kalibrerar sensorn, initialiserar konfigurationsfilen, skriver ut konfigurations variabler om debug variabel är true och startar sampling av barometervärden.

Funktion: **void BMP180_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar I^2C och kalibrerar barometern.

Funktion: **void cfg_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Läser in konfigurationsvariabler från konfigurationsfilen till globala variabler.

Funktion: **void printOut();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver ut konfigurationsvariabler från BMP180 avsnittet i konfigurationfilen om debug variabeln är true.

Funktion: **void connectFifos();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar kommunikationen mellan barometern och SernsorFusion via pipes.

Funktion: **void calibrate();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Läser in kalibreringsvariabler från kalibreringsregister, lagrar de i globala variabler för användning i andra funktioner samt lagrar de i konfigurations filen.

Funktion: **void sample();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Anropar checkPipe och uppdaterar de senaste mätvärdet tills programmet avslutas.

Funktion: **double movInga.vg(float newvalue);**
Returnerar: doubble, filtrerat mätvärde.
Parametrar: float, nytt mätvärde.
Utför: Implementation av ett moving average filter med 10st mötvärden.

Funktion: **void checkPipe();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Kollar ifall en request ligger i pipen och om så är fallet annropar writeOutput som returnerar ett mätvärde.

Funktion: **void writeOutput();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver aktuellt mätvärde till pipen.

Funktion: **void Write(int address, int data);**
Returnerar: Inget.
Parametrar: int address, adress att skriva till. int data, data att skriva till adressen.
Utför: Skriver angiven data till den angivna adressen.

Funktion: **uint8_t Read(int address);**
Returnerar: unsigd int, avläst värde.
Parametrar: int address, adress att lösa från.
Utför: löser av data från den angivna adressen.

Funktion: **uint8_t SetResolution(uint8_t sampleResolution, bool oversample);**
Returnerar: unsigd int, avläst värde.
Parametrar: unsignd int sampelResolution, inställning för mätnogranhet (0-3, där 0 innebär snabbare mätningar med sämre precision och 3 innebär långsammare mätningar med högre precision). bool oversampel, inställning för intern filtrering i barometern.
Utför: Skriver inställningar till barometern.

Funktion: **int GetUncompensatedTemperature();**
Returnerar: int, avläst okompenserad temperatur från barometern.
Parametrar: Inga.
Utför: Instruerar barometern att utföra en mätning, väntar på att mätning genomförs och returnerar okompenserad temperatur.

Funktion: **long GetUncompensatedPressure();**
Returnerar: long, avläst okompenserat tryck från barometern.
Parametrar: Inga.
Utför: Instruerar barometern att utföra en mätning, väntar på att mätning genomförs och Returnerar den okompenserat tryck.

Funktion: **float CompensateTemperature(int uncompensatedTemperature);**
Returnerar: float, kompenserad temperatur.
Parametrar: int okompenserad temperatur.
Utför: använder barometerns unika kalibrerings konstanter för att kompensera en avläst okompenserad temperatur.

Funktion: **float GetTemperature();**
Returnerar: float, kompenserad temperatur.
Parametrar: Inga.
Utför: En mätning av temperaturen med barometern och kompenserar mötvärdet.

Funktion: **long CompensatePressure(long uncompensatedPressure);**
Returnerar: long, kompenserat tryck.
Parametrar: Inga.
Utför: använder barometerns unika kalibrerings konstanter för att kompensera ett avläst okompenserat tryck

Funktion: **long GetPressure();**
Returnerar: long, kompenserat tryck.
Parametrar: Inga.
Utför: En mätning av tryck med barometern och kompenserar mötvärdet.

Funktion: **float GetAltitude(float currentSeaLevelPressureInPa);**
Returnerar: long, akuell höjd över havsnivån.
Parametrar: float currentSeaLevelPressureInPa, nuvarade tryck vid havsnivån.
Utför: Mätningar av både tryck och temperatur och beräknar höjden enligt datablad.

HMC5883L

Funktion: **void main();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Huvudfunktion, initialiserar I^2C, initialiserar konfigurationsfil, skriver ut konfigurations variabler om debug variabeln är true och startar sampling av magnetometern.

Funktion: **void HMC5883L_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar I^2C och skriver inställningar till magnetometern.

Funktion: **void cfg_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Läser in konfigurationsvariabler från konfigurationsfilen till globala variabler.

Funktion: **void printOut();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver ut variabelvärden från HMC588L avsnittet i konfigurationfilen om debug variabeln är true.

Funktion: **void connectFifos();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar kommunikationen mellan magnetometern och SernsorFusion via pipes.

Funktion: **void checkPipe();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Kollar ifall en request ligger i pipen och om så är fallet annropar writeOutput som returnerar aktuellt mätvärde.

Funktion: **void writeOutput();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver aktuellt mätvärde till pipen.

Funktion: **int GetX();**
Returnerar: int, magnetfältets styrka i x-led.
Parametrar: Inga.
Utför: Mätning av magnetfältets styrka i x-led.

Funktion: **int GetY();**
Returnerar: int, magnetfältets styrka i y-led.
Parametrar: Inga.
Utför: Mätning av magnetfältets styrka i y-led.

Funktion: **int GetZ();**
Returnerar: int, magnetfältets styrka i z-led.
Parametrar: Inga.
Utför: Mätning av magnetfältets styrka i z-led.

Funktion: **double computeHeading(int x, int y, int z);**
Returnerar: double, kompassriktning i grader (0-360°, 0°är nord).
Parametrar: int x, int y, int z. Magnetfältets styrka i respektive riktning.
Utför: Beräknar kompassvinkel från magnetfältet styrka i respektive riktning.

Funktion: **void sample();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Anropar checkPipe och uppdaterar kompassriktning tills programmet avslutas.

Funktion: **void calibrate();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Kalibrering av hard iron fel hos magnetometern [46], samt lagrar de senaste kalibreringsvärdena i konfigurationsfilen.

HC-SR04

Funktion: **void main();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Huvudfunktion, initialiserar GPIO och startar sampling av ultraljudssensorn.

Funktion: **HC-SR04_init()**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar GPIO pinar som används av ultraljudssensorn samt anropar connectFifos.

Funktion: **void sample();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Anropar checkPipe och uppdaterar avståndet tills programmet avslutas.

Funktion: **double movingAvg(float newvalue);**
Returnerar: double, filtrerat mätvärde.
Parametrar: float, nytt mätvärde.
Utför: Implementation av ett moving average filter med 10st mötvärden.

Funktion: **int getCM();**
Returnerar: double, int mätvärde.
Parametrar: Inga.
Utför: filterera mätdata på 10 senaset mätpunkterna och Returnerar filtrerat avstånd.

Funktion: **void connectFifos();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar kommunikationen mellan ultraljudssensorn och SernsorFusion via pipes.

Funktion: **void checkPipe();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Kollar ifall en request ligger i pipen och om så är fallet annropar writeOutput som returnerar aktuellt mätvärde.

Funktion: **void writeOutput();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver aktuellt mätvärde till pipen.

Funktion: **int getUltra();**
Returnerar: int, avstånd.
Parametrar: Inga.
Utför: Mäter avståndet och förkaster de värden som övertiger 4 meter.

AutoQuad

Inkluderar:
SensorFusion.c
FlightControl.c
testCases.c

Funktion: **void main();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Startar BMP180, HMC588L och HC-SR04 som bakrundsprocesser, initialiserar sensorFusion samt tar hand om CLI.

SensorFusion

Funktion: **void sfinit();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Annropar InitPipes och sätter startvärde för barometern.

Funktion: **void InitPipes();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Initialiserar kommunikationen mellan sensorprogrammen och SensorFusion.

Funktion: **double getHeight(double uh, double bh);**
Returnerar: double, höjd.
Parametrar: double uh, höjdvärde från ultraljudssensorn. double bh, höjdvärde från barometern.
Utför: Tar in höjd från både barometer och ultraljudsensor och returnerar den mest tillförlitliga höjden.

Funktion: **double getBHeight();**
Returnerar: double, höjd från barometer.
Parametrar: Inga.
Utför: Begär en höjdmätning från BMMP180 processen.

Funktion: **double getUHeight();**
Returnerar: double, höjd från ultralhudssensorn.
Parametrar: Inga.
Utför: Begär en höjdmätning från HC-SR04 processen.

Funktion: **double getHeading();**
Returnerar: double, kompassriktning.
Parametrar: Inga.
Utför: Begär en kompassriktning från HMC588L processen.

Funktion: **void refreshGPS();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Begär en uppdatering av GPS-data och lagrar den i en intern global variabel.

Funktion: **double latitude();**
Returnerar: double, aktuell latitud.
Parametrar: Inga.
Utför: Returnerar aktuell latitud.

Funktion: **double longitude();**
Returnerar: double, aktuell longitud.
Parametrar: Inga.
Utför: Returnerar aktuell longitud.

Funktion: **int quality();**
Returnerar: int, kvalitén hos den aktuella GPS-datan.
Parametrar: Inga.
Utför: Returnerar kvalitén hos den aktuella GPS-datan.

Funktion: **int nsat();**
Returnerar: int, antal satelliter hos den aktuella GPS-datan.
Parametrar: Inga.
Utför: Returnerar antal satelliter hos den aktuella GPS-datan.

Funktion: **double sdn();**
Returnerar: double, aktuell standardavvikelse i nordlig led.
Parametrar: Inga.
Utför: Returnerar aktuell standardavvikelse i nordlig led.

Funktion: **double sde();**
Returnerar: double, aktuell standardavvikelse i östlig led.
Parametrar: Inga.
Utför: Returnerar aktuell standardavvikelse i östlig led.

Funktion: **void commandSensor(char * sensor, char * command);**
Returnerar: Inget.
Parametrar: char * sensor, sensor att skicka komand till. char * command, kommand att skicka.
Utför: Funktion för att skicka komand till andra processer.

Funktion: **void updateLog();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Funktion för att logga sensordata till en textfil.

FlightControl

Funktion: **void PID_cfg_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Läser in variabelvärden från konfigurationsfilen till globala variabler.

Funktion: **void PID_cfg_print();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver ut variabelvärden från PID avsnittet i konfigurationfilen om debug variabeln är true.

Funktion: **void Set_Servo(int num, int pos);**
Returnerar: Inget.
Parametrar: int num, vilket utgång som ska regleras (1 = Aileron, 2 = Elevation, 3 = Thrust, 4 = Rudder). int pos, utslaget utgången ska sättas till (0-100%).
Utför: Sätter en angiven procentsats på på en angiven utgång kopplade till flygkontrollern.

Funktion: **void Arm_FlightController();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Anropar Set_servo i rätt sekvens för att ställa flygkontrollern i ”Armed” läge.

Funktion: **void Disarm_FlightController();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Anropar Set_servo i rätt sekvens för att ställa flygkontrollern i ”Disarmed” läge.

Funktion: **float PIDcal(float diff);**
Returnerar: float, ny styrsignal.
Parametrar: float diff, differensen mellan referenssignal och aktuell position.
Utför: Funktion som beräknar styrsignal med hjälp av en PID-regulator.

testCases

Funktion: **test_cfg_init();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Läser in variabelvärden från konfigurationsfilen till globala variabler.

Funktion: **void test_cfg_print();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Skriver ut variabelvärden från test avsnittet i konfigurationfilen om debug variabeln är true.

Funktion: **void get_time();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Hämtar aktuell tid och lagrar i en globar variabel.

Funktion: **void setHover();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Sätter Thrust till den av konfigurationsfilen bestämda HoverOffset.

Funktion: **void testHoverToStep();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test som börjar med Thrust på värdet av konfigurationsfilen bestämda HoverOffset för att sedan sätta Thrust till värdert av konfigurationsfilen bestämda ThrottleStep samt logga förloppet i en loggfil.

Funktion: **void testRotation();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test som börjar med Thrust på värdet av konfigurationsfilen bestämda HoverOffset för att sedan sätta Rudder till värdert av konfigurationsfilen bestämda 50+YawSpeed följt av 50-YawSpeed samt logga förloppet i en loggfil.

Funktion: **void pitchTest();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test som börjar med Thrust på värdet av konfigurationsfilen bestämda HoverOffset för att sedan sätta Aileron till värdert av konfigurationsfilen bestämda 50+PitchSpeed följt av 50-PitchSpeed samt logga förloppet i en loggfil.

Funktion: **void rollTest();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test som börjar med Thrust på värdet av konfigurationsfilen bestämda HoverOffset för att sedan sätta Elevator till värdert av konfigurationsfilen bestämda 50+RollSpeed följt av 50-RollSpeed samt logga förloppet i en loggfil.

Funktion: **void pidHeightTest();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test för den implementerade höjdregleringen med den av konfigurationfilen bestämda referensen refHeightOne.

Funktion: **void pidHeightTestTwo();**
Returnerar: Inget.
Parametrar: Inga.
Utför: Test för den implementerade höjdregleringen med en första referens den av konfigurationfilen bestämda refHeightOne i 20 sekunder för att i de följande 20 sekundrarna använda refHeightTwo för att slutligen använda refHeightOne i 20 sekunder.

C

Appendix - RTKLIB, konfuigrationsfiler

Uppstartskommandon för RTKLIB

ubx_raw_1hz.cmd
!UBX CFG-RATE 1000 1 1
!UBX CFG-PRT 1 0 0 2256 115200 7 7 0
!UBX CFG-MSG 2 16 0 1 0 1 0 0
!UBX CFG-MSG 2 17 0 1 0 1 0 0

@
!UBX CFG-RATE 1000 1 1

ubx_raw_5hz.cmd
!UBX CFG-RATE 200 1 1

!UBX CFG-PRT 1 0 0 2256 115200 7 7 0
!UBX CFG-MSG 2 16 0 1 0 1 0 0
!UBX CFG-MSG 2 17 0 1 0 1 0 0

@
!UBX CFG-RATE 1000 1 1

ubx_raw_10hz.cmd
!UBX CFG-RATE 100 1 1
!UBX CFG-PRT 1 0 0 2256 115200 7 7 0
!UBX CFG-MSG 2 16 0 1 0 1 0 0
!UBX CFG-MSG 2 17 0 1 0 1 0 0

@
!UBX CFG-RATE 1000 1 1