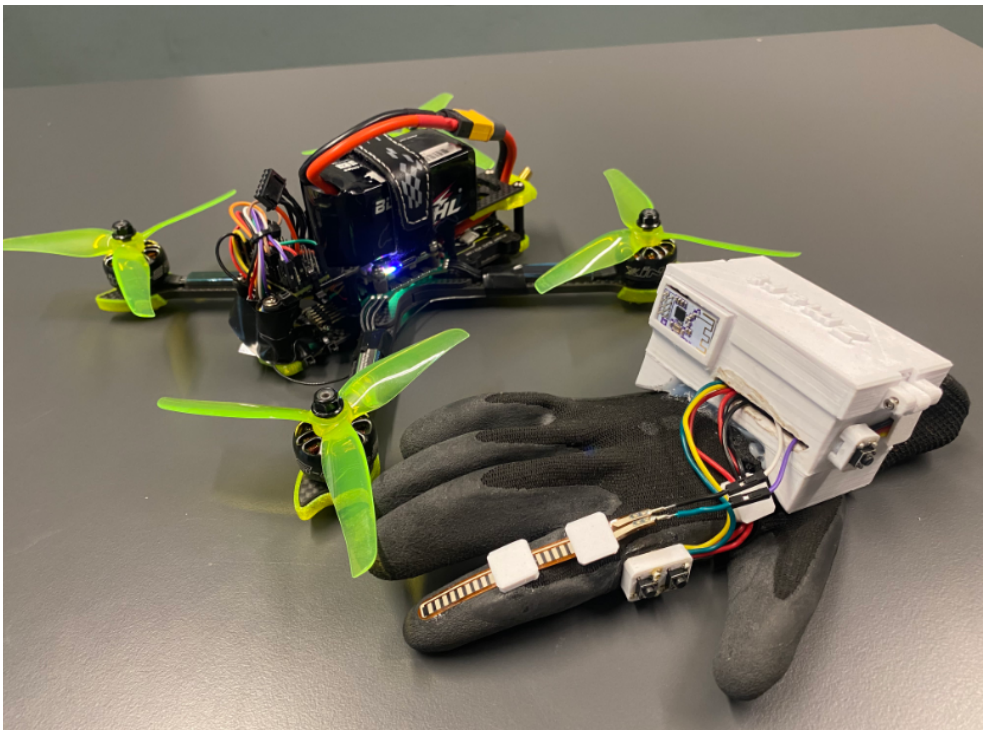
CHALMERS



Framställning av en handstyrd gyroskopisk drönarkontroller

Design och konstruktionsprocess för en gyroskopisk drönarkontroller i form av en handske

Caspian Carlmark

Philip Ferdinandsson

Institutionen för Elektroteknik

CHALMERS TEKNISKA HÖGSKOLA

Göteborg, Sverige 2023

EXAMENSARBETE INOM MEKATRONIK 2023

Framställning av en handstyrd gyroskopisk drönarkontroller

Design och konstruktionsprocess för en gyroskopisk drönarkontroller
i form av en handske

Caspian Carlmark
Philip Ferdinandsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2023

Framställning av en handstyrd gyroskopisk drönarkontroller

Design och konstruktionsprocess för en gyroskopisk drönarkontroller i form av en handske

CASPIAN CARLMARK
PHILIP FERDINANDSSON

Handledare: Carl Kylin, Institutionen för Elektroteknik
Examinator: Erik Agrell, Institutionen för Elektroteknik

Institutionen för Elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Sverige
Telefon: +46 (0)31-772 1000

Förstasida: Bild tagen på färdig kontroller samt drönare, Göteborg 23-06-05

Förord

Detta examensarbete är utfört på institutionen för elektroteknik vid Chalmers tekniska högskola av två studenter på utbildningen Mekatronik, högskoleingenjör. Examensarbetet omfattar 15 högskolepoäng och arbetet har utförts åt konsultföretaget Zmart Engineering under våren 2023.

Vi vill tacka alla som har varit inblandade och gjort detta arbetet möjligt. Stort tack till vår examinator Erik Agrell och vår handledare Carl Kylin för god feedback och hjälp med utvecklingen av denna rapport. Vi vill även tacka Magnus Lorentsen och Edvin Mellberg från Zmart Engineering för deras stöd och tillförande av nödvändig kunskap och material för att lyckas med arbetet.

Sammanfattning

Under de senaste åren har den ökande populariteten för drönare framhävt behovet av en användarvänlig och intuitiv kontrollmekanism. Genom att undersöka styrningen av drönare från handhållna enheter med spakar och knappar till något som utgår från kroppens rörelser, kan det finnas en annan lösning som sänker inträdesbarriären för nya användare.
Detta projekt beskriver designen och funktionaliteten hos en sådan lösning, en gyroskopisk handkontroll för en drönare. För att styra en drönare måste kontrollern kommunicera nödvändig information om throttle, pitch, roll och yaw till drönaren. Denna kommunikation måste ha en pålitlig anslutning för att minska risken för katastrofala fel. Med hjälp av en MPU6050-gyroskop/accelerometer för att mäta handens pitch och roll, knappar för yaw och slutligen en flexsensor som är fäst vid pekfingret för att mäta throttle, mäts all nödvändig information. Denna information sammanställs i en Teensy 4.0-mikrokontroller och skickas över till drönaren genom ett par nRF2401L-sändare/mottagare och tolkas på drönaren med ytterligare en Teensy 4.0-mikrokontroller. Av säkerhetsskäl så är kontrollen utrustad med en knapp för att aktivera och avaktivera drönaren.
För att montera allt på handsken designades ett 3D-utskrivet hölje för komponenterna samt ett knapphölje för de nödvändiga knapparna.
Resultatet av projektet var en fungerande handkontroller som gör att drönarens rörelser efterliknar användarens handrörelser. Vissa problem med lödningen av knapparna gjorde dem något opålitliga, men de fungerade tillräckligt bra för att kontrollen skulle fungera som avsett. För framtida utveckling kan kontrollen lägga till ytterligare funktioner i form av programmerade tecken eller extra spakar/knappar.

Abstract

In recent years, the increasing popularity of drones has highlighted the need for a user-friendly and intuitive control mechanism. By looking differently at the way we control drones from handheld devices with levers and buttons to something that draws from the movement of the body there might be another solution that lowers the barrier of entry for beginners.

This project outlines the design and functionality of such a solution, a gyroscopic hand controller for a drone. To control a drone the controller must accurately communicate the needed information of throttle, pitch, roll and yaw to the quadcopter. This communication must have a reliable connection to reduce the chance of catastrophic failure. With the use of a MPU6050-gyroscope/accelerometer to measure pitch and roll of the hand, as well as buttons for yaw and lastly a flexsensor attached to the index finger to measure throttle all of the needed information is measured. This information gets compiled inside of a teensy 4.0 microcontroller and sent over to the drone through a pair of nRF2401L transceivers and finally interpreted at the drone with another teensy 4.0 microcontroller. For safety the controller is equipped with a button to activate and deactivate the drone.

To make everything attach to the base glove a 3d printed housing for the components was designed as well as a button housing for the needed buttons.

The result of the project was a functioning handcontroller that allows the drone to mimic the movements of the user's hand. Some issues with the soldering of the buttons made them slightly unreliable but functioning enough to make the controller function as intended. For future development the controller could add further functionalities in the form of programmed signs or additional levers/buttons.

Innehållsförteckning

Förkortningar

CRC:	Cyclic Redundancy Check
DLPF:	Digital Lowpass-Filter
DSSS:	Direct-sequence Spread Spectrum
ESB:	Enhanced ShockBurst
ESC:	Electronic Speed Controller
FHSS:	Frequency-Hopping Spread Spectrum
FSK:	Frequency Shift Keying
GFSK:	Gaussian Frequency Shift Keying
I2C:	Inter-Integrated Circuit
IDE:	Integrated Development Environment
IMU:	Inertial Measurement Unit
ISM:	Industrial, Scientific and Medical
MEMS:	Micro-Electro-Mechanical System
MCU:	Microcontroller Unit
SPI:	Serial Peripheral Interface
UART:	Universal Asynchronous Receiver/Transmitter
PID:	Proportional – Integral – Derivative

1. Inledning

Den här rapporten beskriver i detalj utvecklingen av en gyroskopisk handkontroll för en drönare som utfördes på uppdrag av Zmart Engineering. Projektet innefattar en omfattande analys och undersökning av olika aspekter, inklusive trådlös kommunikation, sensorer och den mätdata som behöver hanteras, samt hur dessa olika komponenter ska samspela för att uppnå önskad funktionalitet.

1.1 Bakgrund

Den senaste tiden har drönares popularitet drastiskt ökat och börjat användas inom fler områden. I dagsläget styrs i stort sett alla drönare med konventionella kontrollers, t.ex, joysticks. Dessa kontrollers är inte intuitiva och nya användare har svårt att styra drönaren exakt som de tänkt. Där föds idén om att skapa en kontroller som är mer intuitiv och som följer användarens handrörelser på ett naturligare och mer intuitivt sätt.

Zmart Engineering är ett nystartat ingenjörskonsultföretag som främst fokuserar på maskinteknik, elektroteknik, mjukvaruutveckling och mekatronik. Företaget värnar om ett samhällsansvar och erbjuder därför examensarbeten och jobb till nyexaminerade ingenjörer för att utvecklas i takt med samhället och främja nytänk.

Detta projekt uppkom som en tanke på att implementera ett alternativt sätt att styra drönare, från folk som inte har stor erfarenhet inom drönarstyrning. Med en begränsad bakgrund i området fanns möjligheterna att utforma projektet enligt vad som passade bäst utan att vara präglad av tidigare upplevelser.

1.2 Syfte

Syftet med arbetet är att undersöka möjligheterna till interaktiv styrning av en drönare och därefter implementera detta genom att skapa en kontroller som kan styra en drönare på ett intuitivt sätt. I detta arbete skapas den intuitiva kontrollern genom att fästa sensorer och en mikrokontroller på en handske, så att drönaren kan styras av handrörelser som liknar drönarens rörelse. Genom att skapa en kontroller med intuitiv styrning kan drönare dels bli

mer tillgängliga och användarvänliga men även skapa en roligare upplevelse vid drönarflygning.

1.3 Avgränsningar

Drönaren som ska styras kommer inte att designas, konstrueras eller modifieras utöver att flygkortet/mottagaren kan behöva bytas för att etablera kommunikationen mellan kontrollern och drönare.
Reglersystemet för drönaren kommer inte att modifieras.

1.4 Precisering av frågeställningen

Det finns många intressanta frågor rörande trådlös kommunikation och seriella protokoll när det gäller effekterna som de medför vid drönarflygning. Det finns även många intressanta frågor kring sensorer och valet av dessa för att konstruera en handkontroller. Därför behövs en avgränsning. De frågor som rapporten kommer svara på är:

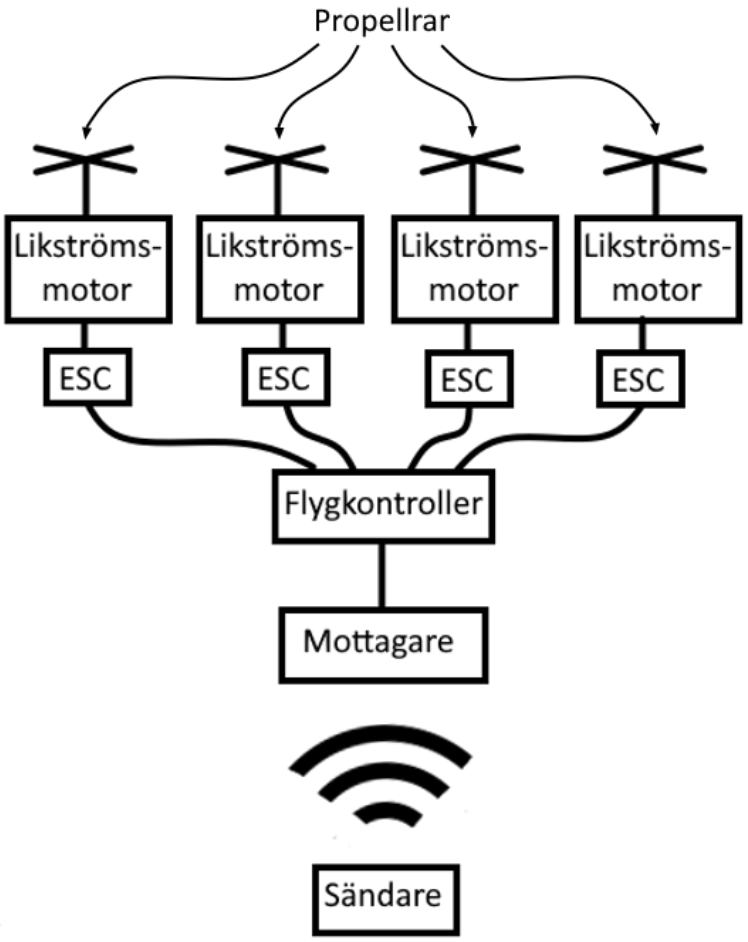
- Vilken av de trådlösa kommunikationsmetoderna: Frequency Shift Keying (FSK), Gaussian Frequency Shift Keying (GFSK), Frequency Hopping Spread Spectrum (FHSS) och Direct-sequence Spread Spectrum (DSSS) är mest lämplig för att styra en drönare med avseende på:
    - säkerhet?
    - räckvidd?
    - snabbhet?
- Hur bör styrsignalerna som skickas till drönaren se ut?
    - Vilken information bör skickas?
    - Vilket format bör användas?
- Vilka sensorer bör användas för att mäta handrörelserna med avseende på:
    - noggrannhet?
    - kostnad?
- Har den designade kontrollern samma funktionalitet som den konventionella kontrollern?

2. Teoretisk / Teknisk bakgrund

Projektet berör flera områden inom bland annat: trådlös kommunikation, kommunikationsbussar, sensorer, och elektroniska komponenter. För att ge bättre kontext så presenteras relevant teoretisk och teknisk bakgrund i detta kapitel.

2.1 Quadcopters uppbyggnad och komponenter

En quadcopter är en obemannad flygfarkost med fyra rotorer. Det finns flera olika modeller av quadcopters som har olika komponenter, men de består av samma grundläggande komponenter vilka kan ses i figur 1. Dessa komponenter är sändaren som skickar signaler trådlöst till mottagaren. Mottagaren skickar vidare dessa signaler till flygkontrollern som skickar induviduella signaler till vardera Electronic Speed Controller (ESC). Varje ESC driver en likströmsmotor där propellrarna är fästa. Inom rapporten används quadcopter och drönare omväxlande.



Figur 1. En quadcopters komponenter och kommunikationsvägen. Sändaren skickar signaler trådlöst till mottagaren som sedan för signalerna vidare till flygkontrollern. Därefter skickar flygkontrollern individuella styrsignaler till vardera ESC som driver likströmsmotorerna där propellrarna är fästa.

2.1.1 Likströmsmotor

På quadcoptern finns det fyra stycken likströmsmotorer vilket kan ses i figur 1. Dessa motorer består av en rotor som roterar i ett magnetfält. Detta magnetfält är skapat av statorn som antingen kan vara en elektromagnet eller vid mindre likströmsmotorer permanentmagnetiserad. Rotorn har poler som är en elektromagnet med radiellt riktat magnetfält och dessa aktiveras genom borstar som för yttre kommande likström över kollektorn. Kollektorn är en skiva eller trumma som har lika många kontaktytor som rotorn har poler. [1]

2.1.2 Electronic Speed Controller

ESC är en elektrisk krets som styr och reglerar en likströmsmotors hastighet. Den tar emot ett börvärde för motorns hastighet i form av en analog signal och ger ut styrsignaler som driver likströmsmotorn. I figur 1 syns att varje ESC tar emot det analoga börvärdet från flygkontrollern och skickar signaler som driver likströmsmotorn. En ESC kan enbart driva den typen av motor som ESCn är byggd för, exempelvis en borstlös likströmsmotor eller en borstad likströmsmotor. [2]

2.1.3 Flygkontroller

Flygkontrollern är den mest centrala komponenten i en quadcopter. Dess huvudsakliga uppgift är att styra och reglera de fyra olika motorerna individuellt utefter börvärden som användaren bestämmer och sensordata från de olika sensorerna som finns på quadcoptern. Den åstadkommer detta genom att kommunicera med mottagaren via en av flygkontrollerns universal asynchronious receiver/transmitter (UART) pins. Därefter matas dessa börvärden tillsammans med de uppmätta värdena in i en Proportional-Integral-Derivative (PID) kontroller som genererar en styrsignal till respektive ESC som driver motorerna vilket kan ses i figur 1. Genom att styra varje motor individuellt möjliggörs styrning av quadcopters rörelse i tre dimensioner. [3]

2.1.4 Mottagare

Mottagaren är en komponent på quadcoptern som möjliggör trådlös kommunikation. Den lyssnar och hämtar information från radiovågor och skickar vidare informationen till flygkontrollern som styr quadcoptern baserat på informationen som mottagaren läst. I figur 1 syns att mottagaren tar emot signaler trådlöst från sändaren och skickar vidare dessa till flygkontrollern. Eftersom funktionaliteten av mottagaren är att kommunicera med olika komponenter är kommunikationsprotokoll en viktig del för mottagaren. Mottagaren måste stödja både det protokollet som sändaren skickar ut radiovågorna med (TX protokoll), samt det protokollet som flygkontrollen förväntar sig att informationen från mottagaren kommer med (RX protokoll). [4]

2.1.5 Sändare

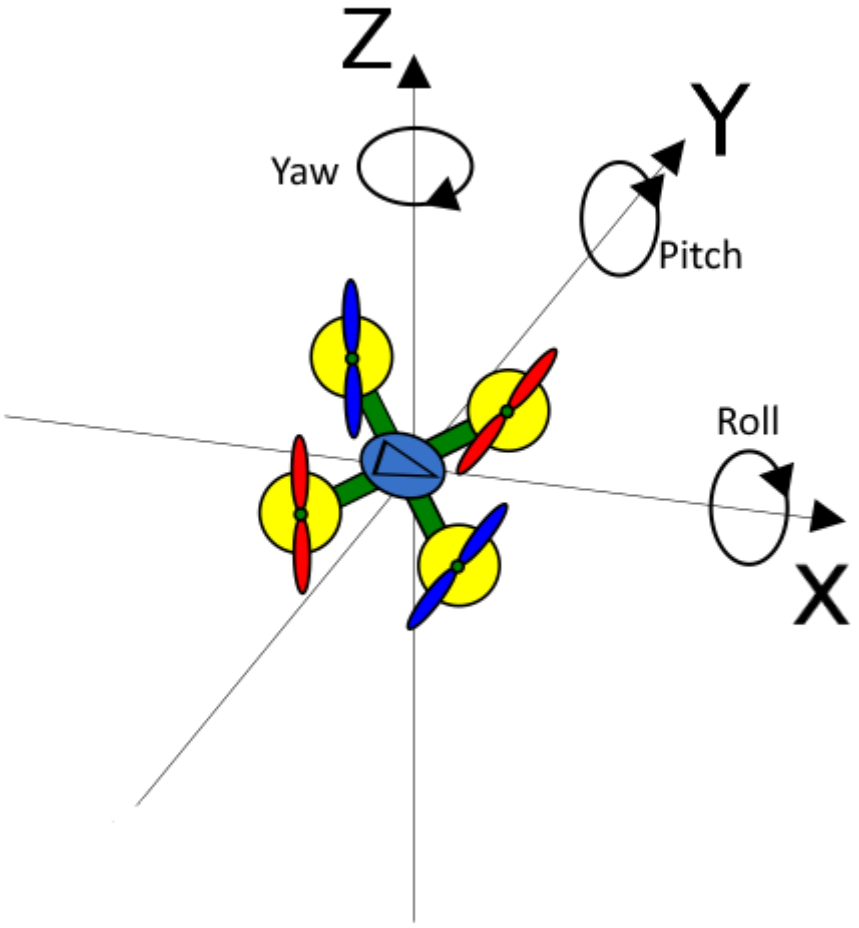
Sändaren är en komponent som inte befinner sig på quadcoptern vilket kan ses i figur 1. Den befinner sig nära användaren på marken. Sändaren är ansvarig för att skicka signaler via

radiovågor som mottagaren kan läsa. Informationen som sändaren skickar är de olika börvärden för quadcoptern, såsom rotationsvinklar och motorernas hastighet. Dessa börvärden bestäms av användaren genom att till exempel styra joysticks och trycka på knappar på kontrollern. Likt mottagaren så är det viktigt att sändaren skickar ut informationen med ett protokoll som mottagaren kan läsa. [4]

2.2 Styrning av quadcopters

En quadcopter har per definition fyra motorer/propellerar och sex frihetsgrader. Tre av dessa frihetsgrader är translationsrörelser längs de horisontella och vertikala axlarna och de resterande tre frihetsgraderna är rotationsrörelser runt samma axlar. En quadcopter har inte förmågan att direkt göra en translationsrörelse i det horisontella planet, utan utnyttjar istället en rotationsrörelse för att vinkla quadcoptern och därmed flytta kraftresultanten som skapas av propellernas rörelse. Detta resulterar i att quadcoptern gör en translationsrörelse i samma riktning som den vinklades.

2.2.1 Benämning av quadcopters rotationsaxlar



Figur 2. Benämning av en quadcopters rotationsaxlar. Originell bild hämtad från [5].

I figur 2 visas axlar och rotationer för en quadcopter i ett tredimensionellt koordinatsystem. I figur 2 ser man att en rotation runt y-axeln, även kallad pitch, resulterar i en translationsrörelse längst x-axeln. Likväl så resulterar en rotation runt x-axeln (roll) i en translationsrörelse längs y-axeln. Quadcoptern kan göra dessa rotationsrörelser genom att kontrollera hastigheten på vardera propeller. För att ändra pitch så har de två främre motorerna olika hastigheter än de bakre. För att ändra roll så har de två högra motorerna en annan hastighet än de vänstra. Vidare så är propellrarnas rotationsriktning olika, i figur 2 så har de röda propellrarna motsatt rotationsriktning mot de blå. Detta leder till att drönaren

även kan rotera runt z-axeln (yaw) genom att ha olika hastigheter på dessa röda och blåa par. Slutligen så kan quadcoptern röra sig längst z-ledet (throttle) genom att öka eller minska hastigheten på samtliga motorer.[6]

2.2.2 Styrsignaler

Det behövs fyra huvudsakliga kanaler/signaler för att styra drönaren. Dessa är börvärden för drönarens rotationsvinklar: roll, pitch och yaw, och för lyftkraften: throttle. När det gäller styrsignalerna för roll och pitch så ska de ha ett värde på 1500 för att den aktuella vinkeln ska vara noll. Ett högre eller mindre värde än detta resulterar i en positiv vinkel (moturs riktad rotation) respektive negativ vinkel (medurs riktad rotation) runt den aktuella axeln sedd framifrån. Till skillnad från roll och pitch där börvärdet ger en konstant vinkel så ger yaw en konstant vinkelhastighet. Ett högre eller mindre värde än 1500 resulterar i en positiv rotation runt z-axeln (moturs) respektive negativ rotation runt z-axeln (medurs).[7]

För att ändra inställningar på drönarens flygkort används programmet BetaFlight Configurator. I många fall så är mottagaren och sändaren inte rätt inställda från början och styrsignalerna skickas på fel kanaler. Detta måste ställas in via BetaFlight Configurator så att en rörelse av joysticksen på kontrollern motsvarar rätt rörelse på drönaren. Vidare så finns en inbyggd funktion i BetaFlight Configurator så att man kan förhandsgranska vilka signaler som mottagaren läser och hur drönaren svarar på dessa signaler.

2.3 Sensorer

För att mäta upp handrörelser används sensorer. I detta avsnitt behandlas teori, uppbyggnad och användningsområden för relevanta sensorer.

2.3.1 Gyroskop

Det finns flera olika typer av gyroskop som använder olika tekniker men i denna rapport är Micro-Electro-Mechanical System (MEMS) gyroskop i fokus. MEMS gyroskop använder sig av ett vibrerande element som till exempel ett piezoelement och utnyttjar coriolis effekten för att mäta corioliskraften, vilken är proportionell till vinkelhastigheten [8]. Genom att sedan integrera mätvärdena på vinkelhastigheten över tid kan man bestämma den aktuella vinkeln kring alla rotationsaxlar. MEMS gyroskop har en tendens till att mätvärdena driver iväg över tiden, vilket gör att vinkelmätningar blir mindre pålitliga.[9]

2.3.2 Accelerometer

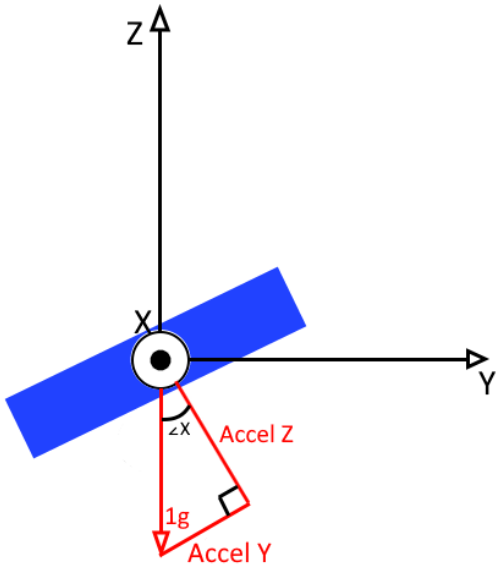
En accelerometer mäter acceleration och lutning genom att mäta kraften som verkar på en provmassa. Den vanligaste typen av accelerometer är MEMS accelerometer. Denna består av en rörlig del och en fast del. När acceleration eller rörelse uppstår rör sig provmassan i förhållande till sensorn och genererar en elektrisk signal [9]. Accelerometrar mäter både statisk och dynamisk acceleration för att mäta lutning, hastighetsförändringar och vibrationer. En nackdel med att använda en accelerometer för att bestämma en vinkel är att den påverkas av yttre krafter. Om accelerometern är plan mot marken (0°) och rör sig horisontellt (så att vinkeln förblir 0°) så påverkas mätvärdena och den beräknade vinkeln blir inkorrekt.

För att beräkna en accelerometers lutning så används geometri och förhållanden mellan de olika axlarnas mätvärden, se figur 3 och 4. För beräkning av vinkeln runt x-axeln, respektive y-axeln används ekvationerna:

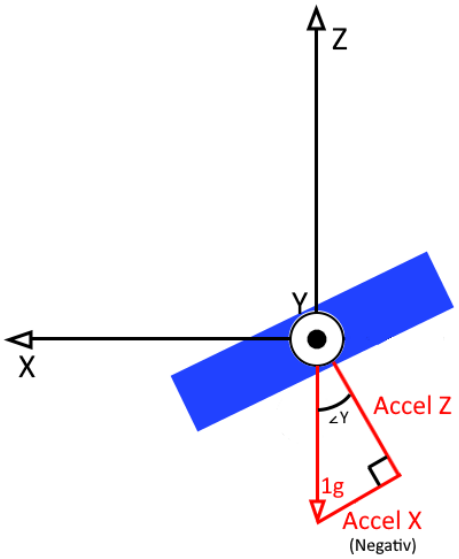
$$\angle x = tan^{-1}\left(\frac{Accel_y}{Accel_z}\right), \qquad (1)$$

$$\angle y = -\ tan^{-1}\left(\frac{Accel_x}{Accel_z}\right), \qquad (2)$$

där $Accel_x$, $Accel_y$ och $Accel_z$ är accelerometers mätvärden för x-axeln, y-axeln respektive z-axeln.



Figur 3. En accelerometers geometri och dess mätvärden för att beräkna lutningen runt x-axeln. Accel Z och Accel Y är accelerometerns mätvärden för z- och y-axlarna.



Figur 4. En accelerometers geometri och dess mätvärden för att beräkna lutning runt y-axeln. Accel Z och Accel X är accelerometerns mätvärden för z- och y-axlarna.

2.3.3 Complementary- och Kalman-Filter

Eftersom både MEMS gyroskop och MEMS accelerometrar har sina nackdelar och är opålitliga då de ska mäta en vinkel individuellt så kan deras mätvärden sammanföras för att den beräknade vinkeln ska bli mer pålitlig. Det är vanligt att MEMS- gyroskop och accelerometrar sätts på samma kort, så kallade Inertial Measurement Units (IMU:s) av denna anledning. För att sammanföra mätvärden från gyroskopet och accelerometern används ofta ett Complementary- eller Kalman-filter. Kalman-filtret är mer robust men också mer komplext och beräkningsintensiv än Complementary-filtret, men båda filter lyckas få noggranna resultat vid vinkelmätning vid både statiska och dynamiska mätningar.[9]

Complementary filtret har formeln:

$$\theta_{angle} = \alpha \cdot \left(\theta_{angle} + \omega_{gyro} \cdot dt\right) + (1 - \alpha) \cdot a_{accel}, \qquad (3)$$

där θ_{angle} är vinkeln, ω_{gyro} är rotationshastigheten läst från gyroskopet, a_{accel} är vinkeln beräknat från accelerometerns data, dt är tiden sedan senaste mätningen och α är filter koefficienten,

$$\alpha = \frac{\tau}{\tau + dt}, \qquad (4)$$

där τ är tidskonstanten för filtret. [9]

2.3.4 MPU6050

MPU6050 är en IMU som inkluderar en processor, ett MEMS gyroskop och en MEMS accelerometer. Den råa mätdatan från gyroskopets tre axlar och accelerometerns tre axlar sparas i dedikerade register inom processorn. Processorns register används dessutom till inställningar för kortet och MEMS sensorerna, t.ex, skalan på mätdatan, konfigurering av Digitalt Lowpass Filter (DLPF) och energiinställningar. För att läsa från eller skriva till MPU6050s register så används Inter-Integrated Circuit (I2C) bussen/protokollet. [10]

2.3.5 Flex Sensor

Flexsensorn är en resistiv sensor vars resistans ändras när den böjs, vilket gör det möjligt att mäta böjning. Genom att koppla jord med en resistor och därefter seriekoppla flexsensorn med resistorn kan man mäta spänningen och därmed hur stor resistansen på flexsensorn är. Flexsensorn består bland annat av ett strömledande bläck, som ökar resistans då bläcket deformeras.[11]

2.4 Kommunikation mellan komponenter samt kontroller och drönare

2.4.1 Transceiver

En transceiver är en elektronisk komponent som kan både sända och ta emot radiosignaler via sin antenn. Fördelen med en komponent som både kan sända och ta emot radiovågor på ett kort är att produktionskostnaden minskar. Transceivers används ofta i tvåvägs radios som, till exempel, walkie-talkies. [12]

2.4.2 nRF2401L

nRF2401L är en tranceiver som utnyttjar det licensfria Industrial, Scientific and Medical (ISM) -bandet på 2,4 GHz. Transceivern använder GFSK modulering samt Enhanced ShockBurst (ESB) protokollet för att kommunicera trådlöst med andra transceivers. Transceivern använder även Serial Peripheral Interface (SPI) bussen/protokollet för att kommunicera med andra enheter såsom en Mikrokontroller enhet (MCU).[13]

2.4.3 Enhanced ShockBurst

ESB är ett kommunikationsprotokoll som används av bland annat transceivern nRF2401L. Paketen som skickas består av [13]:

- 1 byte inledning
    - Antingen 01010101 eller 10101010. Denna byte är till för att synkronisera den mottagande transceivern med den sändande transceivern och resten av paketet
- 3-5 bytes adress
    - Den mottagande transceivern läser enbart resten av paketet om paketets adress stämmer överens med den mottagande transceivers adress.
- 9 bitar pakets kontroll fält där:
    - 6 bitar används för att representera datans längd i bytes.
    - 2 bitar används för paket identifiering. Dessa anger om paketet är helt nytt eller om paketet har försökts skickas på nytt.
    - 1 bit som används för att meddela den mottagande transceivern om den ska skicka en bekräftelse att paketet tagits emot eller inte, en etta på denna bit innebär att den mottagande transceivern inte ska bekräfta paketet, en nolla innebär att den ska skicka tillbaka en bekräftelse.
- 0-32 bytes data
    - Datan som ska överföras mellan enheterna.
- 1-2 bytes Cyclic Redundancy Check
    - Cyclic Redundancy Check (CRC) genererar ett polynom baserat på innehållet i paketet. När den mottagande transceivern har läst innehållet i paketet genererar den ett polynom baserat på samma innehåll. Om det polynom som den mottagande transceivern genererat inte är samma som det polynom i paketet så innebär det att paketets innehåll har ändrats, och därför accepteras inte paketet.

2.4.4 SBUS

SBUS är ett digitalt kommunikationsprotokoll som används inom remote control (RC)-branschen för att överföra styrsignaler mellan mottagare och flygkontrollern. Protokollet använder en enda digital signal för att överföra upp till 16 kanaler av styrdata, tillsammans med ytterligare data såsom fail-safe inställningar och telemetridata.[14]
Ett SBUS paket är 25 bytes lång och innehåller:

- Byte[0]: inledning, 0x0F
- Byte[1-22]: 16 servo kanaler, 11 bitar per kanal
- Byte[23]
    - Bit 0: kanal 17 (0x01)
    - Bit 1: kanal 18 (0x02)
    - Bit 2: uppkoppling tappad-flagga (0x04)
    - Bit 3: automatisk nödlandning (0x08)
- Byte[24]: avslutning 0x00

2.4.5 Inter-Integrated Circuit

I2C-protokollet är ett synkront-protokoll och består av en seriell-datalinje och en seriell-klocklinje. Varje sekundärnod (t.ex en MPU6050) har en 7 bitars adress, vilket innebär att upp till 128 sekundärnoder kan befinna sig på samma linje [15]. I projektet används en MPU6050 och dess standardadress är det hexadecimala talet 0x68.

2.4.6 Serial Peripheral Interface

SPI bussen används av en CPU/MCU för att kommunicera med andra sekundära enheter. Bussen består av en seriell-klocklinje, en datalinje för data från den primära enheten till den sekundära enheten, en datalinje för data från den sekundära enheten till den primära enheten samt en aktiveringssignal för varje sekundärenhet. Eftersom det finns två olika linjer med data, som dessutom går åt olika håll så är det möjligt att överföra data åt båda hållen samtidigt [16]. I projektet används tranceivern nRF2401L som använder SPI för att kommunicera med mikrokontrollern.

2.5 Mikrokontroller

En MCU är en integrerad krets som är designad för att styra specifika enheter eller utföra specifika funktioner. Den består av en processor, minne samt in/ut-portar på en enda krets. Mikrokontrollern tar emot inmatning från sensorer eller andra enheter, bearbetar denna information och styr sedan utmatning till andra enheter, såsom motorer eller displayer.[17]

2.5.1 Teensy 4.0

Teensy 4.0 är en MCU inkluderande bland annat en ARM Cortex-M7 processor, 1984 kB flash-minne, 1024 kB RAM-minne och totalt 40 in- och ut-portar. Av dessa portar så finns det

7 par som kan hantera seriell kommunikation och dataöverföring samt 3 olika SPI bussar och 3 olika I2C bussar. Teensy 4.0 är kompatibel med Arduino Integrated Development Environment (IDE) vilket gör det enkelt att programmera och ladda upp kod till MCU:n via en USB sladd.[18]

2.6 Trådlös kommunikation

Alla moderna trådlösa kommunikationsmetoder bygger på elektromagnetisk strålning, i synnerhet radiovågor (undantag infraröd). Det som skiljer metoderna åt är hur informationen skickas och tolkas via dessa vågor (modulation), vad datan som skickas har för format och i vilken ordning det kommer (protokoll) och vilket frekvensområde som används. Det finns många olika modulationstekniker men i tabell I listas några grundläggande och populära alternativ.

TABELL I

Några grundläggande och populära modulationstekniker

Namn	Förkortning	Typ	Förklaring
Amplitudmodulation	AM	Analog	Amplituden på den bärande vågen ändras proportionellt mot datan som skickas. [19]
Frekvensmodulation	FM	Analog	Frekvensen på den bärande vågen förändras proportionellt mot datan som skickas. [19]
Frequency Shift Keying	FSK	Digital	Den bärande vågen skiftar mellan två olika frekvenser, där den ena frekvensen representerar en logisk etta och den andra frekvensen en logisk nolla. [20]
Gaussian Frequency Shift Keying	GFSK	Digital	Samma princip som FSK men med ett gaussiskt filter på signalen för att undvika “spill” på andra frekvenser. [21]
Direct-Sequence-Spread Spectrum	DSSS	Spread Spectrum	Sprider ut vågor som representerar digitala signaler över ett bredare band. [19]
Frequency Hopping Spread Spectrum	FHSS	Spread Spectrum	En digitalt representerad signal hoppar mellan pseudo-slumpmässiga frekvenser. [22]

2.6.1 Störningar och Säkerhet

Av de nämnda modulationsteknikerna i tabell I, så har DSSS och FHSS bäst förutsättningar för att motverka störningar. Dessa metoder är i grunden skapade för att minska störningars påverkan. DSSS gör detta genom att sprida ut sig på ett bredare spektrum, så att störningar på enskilda frekvenser inte har lika stor inverkan [19]. En ytterligare säkerhetsåtgärd som DSSS använder är så kallad Chipping. Resultatet är att signalen blir krypterad, vilket gör det svårt för externa lyssnare att dekryptera om de inte har tillgång till chipping-koden. FHSS å andra sidan, byter regelbundet frekvensen som används och kan därmed hålla sig undan de frekvenser där störningar förekommer, vilket även gör det svårt för externa lyssnare att läsa av datan om de inte känner till vilka frekvenser som används[19, 23]. AM, FM, FSK och

GFSK är låsta till sina förutbestämda frekvenser och har inte någon direkt möjlighet att undvika störningar som befinner sig på samma frekvenser.

2.6.2 Räckvidd

En annan aspekt som kan påverka säkerheten på liknande sätt som störningar är räckvidden. Om mottagaren kommer utanför sändarens räckvidd eller om signalen blir för svag så kan liknande fenomen hända som när störningar inträffar. Den maximala räckvidden beror på den minimala signalstyrkan mottagaren klarar av och effekten som sändaren skickar ut signaler med. Signalstyrkan avtar ju längre bort mottagaren är från sändaren och följer sambandet:

$$S = \frac{P_t}{4\pi d^2}, \qquad (5)$$

där S är signalstyrkan, P_t är sändarens effekt och d är avståndet mellan sändaren och mottagaren [24]. Den teoretiska maximala räckvidden fås därför genom att lösa ut avståndet från (5) samt att sätta in den minimala signalstyrkan som mottagaren kräver som S:

$$d = \sqrt{\frac{P_t}{4\pi S}}. \quad (6)$$

Från (6) så syns det att den maximala räckvidden ökar då sändarens effekt (P_t) ökar samt om en känsligare mottagare används (S minskar). Den minimala signalstyrkan S beror på vilket signal-till-brus-förhållande som är det minsta kravet för mottagaren som används, vilket i sin tur beror på modulationstekniken. Det innebär att både modulationstekniken och störningar har en påverkan på räckvidden och att modulationstekniker som är mer tåliga mot störningar har en längre räckvidd.

Eftersom drönaren ska användas i Sverige så behöver svenska lagar tas i beaktning. Kommunikationen behöver ske på ett licensfritt band, såsom ISM-bandet på 2,4 GHz, där den utsända effekten inte får överstiga 25 mW för ospecificerade tillämpningsområden [25].

3. Metod

För att besvara frågeställningarna och konstruera en handkontroller har arbetet delats upp i 5 steg, dessa steg är:

- Val av kommunikationsmetod
- Undersökning av styrsignaler
- Design och val av komponenter
- Programmering och konstruktion av handkontrollern
- Funktionalitetstestning, felsökning och optimering.

3.1 Val av kommunikationsmetod

För att besvara vilken kommunikationsmetod som är mest lämplig att använda för en drönare utifrån frågeställningen gjordes en litteraturstudie om de olika kommunikationsmetoderna för att få en uppfattning om hur de fungerar och vilka för- och nackdelar varje metod har. Efter att information om de olika kommunikationsmetoderna samlats in jämfördes de mot varandra utifrån punkerna som tas upp i frågeställningen. Efter en avvägning valdes en kommunikationsmetod som ska användas i projektet.

3.2 Undersökning av styrsignaler

För att ta reda på vilken information drönaren behöver för att kunna flyga granskades “Receiver tab” i BetaFlight Configurator. Där hittades alla nödvändiga signaler och på vilken kanal de skickades samt hur de ska vara representerade. I “Receiver tab” hittades även vilka seriella protokoll som kan användas för kommunikation mellan flygkontrollern och mottagaren. Av valmöjligheterna fanns SBUS vilket valdes då det underlättade i kopplingen genom att bara kräva en port istället för flertalet separata sladdar.

3.3 Design och val av komponenter

Valet av de olika komponenterna genomfördes genom en avvägning av funktionaliteten som önskades för kontrollern, samt kostnad och tillgänglighet av olika alternativ. Med hänsyn till att komponenten var både kostnadseffektiv och väldokumenterad, valdes MPU6050 som gyroskop/accelerometer för kontrollern. Det fanns flera alternativ att välja mellan för mikrokontrollern, såsom Arduino Nano och Teensy 3.5. Valet av Teensy 4.0 baserades på att det betraktades som en kompetent och prisvärd komponent enligt projektets handledare.

Flexsensor valdes för att den passade in i funktionalitetsmålet att styra drönarens rörelser genom handrörelser.

När komponentvalen var gjorda var det ett krav att de skulle monteras på kontrollern på ett kompakt och skyddat sätt för att minimera risken för yttre påverkan genom stötar och slag. Detta problem löstes genom att designa strukturer som 3D-printades. Dessa strukturer konstruerades för att hålla komponenter och kablar på plats.

3.4 Programmering och konstruktion av handkontrollern

Första steget för att programmera MCU:n som ska utgöra en del av handkontrollern är att läsa produktbladen för de valda komponenterna. Sen behöver de fysiska kopplingarna mellan komponenterna och MCU:n göras baserat på informationen från produktbladen och MCU:ns olika egenskaper för separata in- och ut-portar. Därefter behöver koden utformas i Arduino IDE eller annan IDE så att mikrokontrollern hämtar data från sensorerna för att bestämma vilka vinklar handen/sensorerna har och övriga sensorer som ska styra andra funktioner. Efter att sensorerna har blivit avlästa behövs dessa mätningar bearbetas och styrsignalerna som ska skickas till drönaren behöver beräknas med hjälp av dessa mätningar. Efter att styrsignalerna har blivit beräknade behövs kod för att skicka dessa till handkontrollerns transceiver och därefter vidare till drönarens mottagare.

För att säkerställa att mottagaren stödjer det valda kommunikationssättet konstruerades en egen mottagare bestående av en transceiver och en MCU. Funktionerna som denna mottagare måste ha är att först läsa in styrsignalerna från transceivern som är kopplad till MCU:n. Därefter behöver MCU:n skicka vidare dessa styrsignaler till drönaren via ett seriellt protokoll som drönaren stödjer. För att den konstruerade mottagaren ska ha denna funktionalitet så behöver MCU:n programmeras. Koden för MCU:n skrivs i Arduino IDE och externa bibliotek används för att underlätta skrivningen av programkoden.

3.5 Funktionalitetstestning, felsökning och optimering

Första steget med att testa funktionalitetet av systemet i helhet är att analysera de olika systemen var för sig. För att testa drönaren och att dess kommunikation funkar så testkörs den med en etablerad kontroller. När testflygning är avslutad tas de olika styrsignalernas värde och analyseras genom att hämta dem från Betaflights Blackbox som läser flygkortets minne

och de sparade styrsignalerna från flygningen. Det andra systemet är handkontrollern där det kontrolleras att signalerna kommuniceras korrekt mellan handkontrollerns sändare och drönarens mottagare. Datan från Blackboxen används sedan för att kalibrera hur stora styrsignalerna ska vara från kontrollern för att drönaren ska ha önskad funktionalitet.

4. Genomförande

4.1 Litteraturstudie

En stor del av litteraturstudien ämnades till att svara på följande frågor:

1. Vilken trådlös kommunikationsmetod är mest lämplig att använda för kommunikation med en drönare med avseende på:
    - säkerhet?
    - räckvidd?
    - snabbhet?
2. Hur bör styrsignalerna som skickas till drönaren se ut?
3. Vilka sensorer bör användas för att mäta handrörelserna med avseende på:
    - noggrannhet?
    - kostnad?

4.1.1 Kommunikationsmetod

När det gäller att välja ett kommunikationssätt lämpat för att styra en drönare från ett säkerhetsperspektiv så är den primära drivaren av valet vilken metod som är mest resistiv mot störningar. Störningar kan påverka kommunikationen mellan kontroller och drönare så att föraren kan tappa kontrollen vilket kan leda till att drönaren störtar, orsakar olyckor eller skador på människor och material. Av de jämförda metoderna så är FHSS mest tolerant mot störningar och eftersom styrsignalerna som skickas till drönaren inte innehåller känslig information som behöver krypteras så är det mest lämpligt att använda FHSS för att kommunicera med en drönare från ett säkerhetsperspektiv.

DSSS är snabbare och kan skicka större mängd data på samma tid som FHSS och andra kommunikationsmetoder kan [23]. Denna skillnaden kan dock vara obetydlig när det kommer till att välja vilken som är mest lämpad till att styra en drönare då informationen som skickas inte har en stor storlek och både FHSS och DSSS passar bra för detta ändamål. Kommunikationens räckvidd är kopplad till den utsända effekten hos sändaren, och vilken modulationsteknik som används. Vissa modulationstekniker är mer tåliga mot brus än andra och behöver inte ett lika högt signal-till-brus-förhållande. Den utsända effekten är lagreglerad

för samtliga modulationstekniker och både DSSS och FHSS är relativt tåliga mot störningar och har en tillräckligt bra räckvidd för att styra en drönare.

Av dessa anledningar så är FHSS och DSSS de kommunikationsmetoder som är mest lämpliga för att styra en drönare. På grund av en missuppfattning så valdes DSSS bort som lämplig modulationsteknik, men skulle i själva verket passa bra för detta projekt. Istället valdes FHSS som mest lämplig modulationsteknik då det var det mest optimala alternativet då detta val gjordes. Men på grund av att FHSS ofta används i kortdistans kommunikation hos bluetooth enheter så är det svårt att införskaffa hårdvara som både använder FHSS och har tillräckligt hög räckvidd för att använda till drönar-kommunikation. Därför, för att ha en tillräckligt hög och rimlig räckvidd för en drönare användes istället transceivern nRF2401L. Transceivern använder GFSK som modulationsteknik i projektet. Detta innebär att kommunikationen löper större risk för att bli påverkad av störningar än om FHSS hade använts. Däremot så använder nRF2401L ESB protokollet där en obligatorisk CRC används. På så sätt kan störningars påverkan minimeras och felaktiga signaler till mottagaren undvikas även om kommunikationen blir påverkad av störningar. En annan fördel med att använda transceivers som t.ex, nRF2401L är att man kan utnyttja tvåvägskommunikation. Tvåvägskommunikation är inte ett krav för att kommunicera med en drönare då det endast krävs att styrsignaler skickas till drönaren för att flyga, men det kan vara fördelaktigt under utveckling av kontrollern. Man kan exempelvis utnyttja den inbyggda bekräftelsefunktionen hos nRF2401L för att enkelt se om kommunikationen är stabil eller om paketen inte når den mottagande transceivern.

4.1.2 Sensorer

För att tolka de styrsignaler som behövs för att styra drönaren på ett korrekt sätt behövs sensorer. Dessa styrsignaler är:

- Throttle
- Yaw
- Pitch
- Roll

Under projektets gång så framkom flera alternativ på hur man ska få fram dessa men de sensorerna som valdes var:

- Flexsensor för throttle

- Knappar för yaw
- MPU6050 för Roll/Pitch

Valet av dessa var en kombination av pris och funktionalitet. Tanken var att roll och pitch skulle kontrolleras av handens/handkontrollerns lutningar, så att drönarens lutningar följer handens lutningar runt de horizontella axlarna. Det fanns alltså ett behov av en sensor som kan mäta vinklar. MPU6050 IMU:n är en mycket populär komponent både på grund av dess låga pris samt att det går att få noggranna vinkelmätningar genom att kombinera MEMS-gyroskopet och accelerometern via ett Complementary- eller Kalman-filter.
Flexsensorn valdes då dess funktion passade in i hur vi hade tänkt oss att kontrollens funktionalitet skulle vara, throttle skulle kontrolleras genom att böja på fingrarna. För yaw valdes knappar då de var till hands. Samma resultat i både funktionalitet samt uppbyggnad kunde skett med en spak, switch eller potentiometer.

Det uppkom alternativa förslag av sensorer som kunde användas istället för MPU6050 och flexssensorn men dessa blev bortvalda av olika anledningar. Man skulle till exempel kunna använda en kamera som mäter upp handens lutningar och fingrets böjning, men då detta är både dyrare och innebär mer komplexitet i kod samt mindre mobilitet än en MPU6050- och flexsensor-kombination så valdes den bort. Istället för en flexssensor skulle en hallsensor på handflatan och en magnet fäst på fingerspetsen kunna mäta fingrets böjning. Men eftersom avståndet som en hallsensor kan mäta med hjälp av små magneter är begränsat så skulle detta innebära att känsligheten för handkontrollern skulle påverkas samt att det skulle uppstå en “död zon” där en viss del av fingrets böjning inte skulle påverka styrsignalen alls.

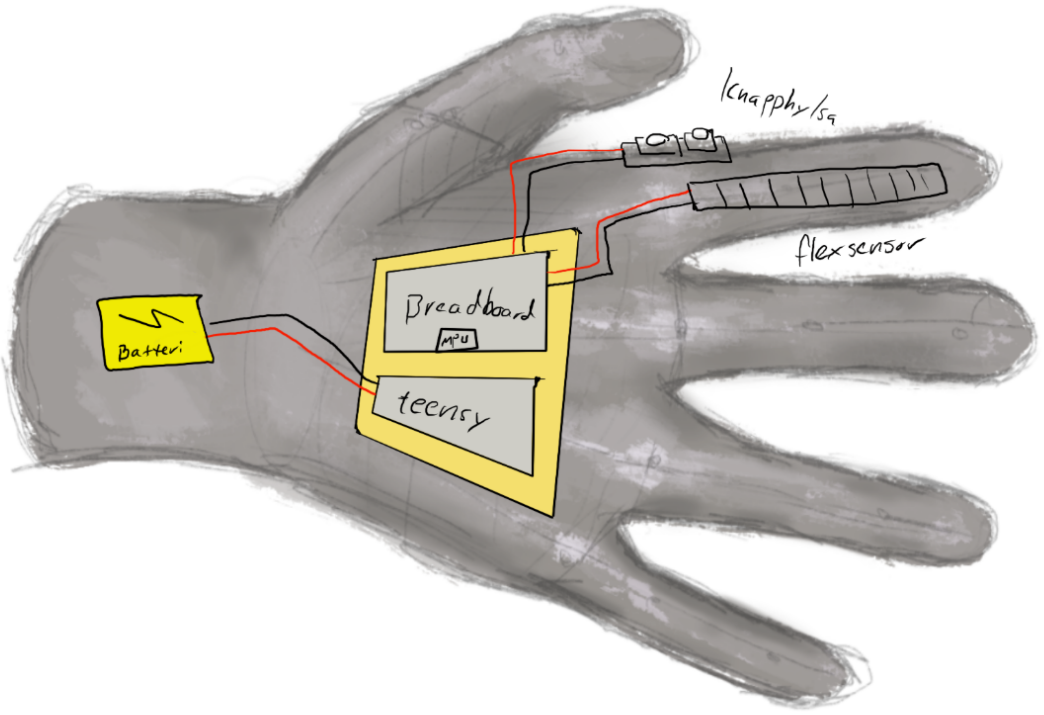
4.2 Design av kontroller

I detta avsnitt behandlas designvalen av handsken och hur den verkliga designen kopplas till den preliminära ritningen. Vidare redovisas val för hur komponenterna placeras för att effektivisera plats och funktionalitet, samt de olika problem som uppkom under designen av kontrollern och hur dessa löstes.

4.2.1 Handsken

Idén för kontrollern var att den skulle vara styrd av handrörelser. Böjning av fingrarna för att bestämma throttle och att drönaren skulle matcha handens position när den vinklas. Det

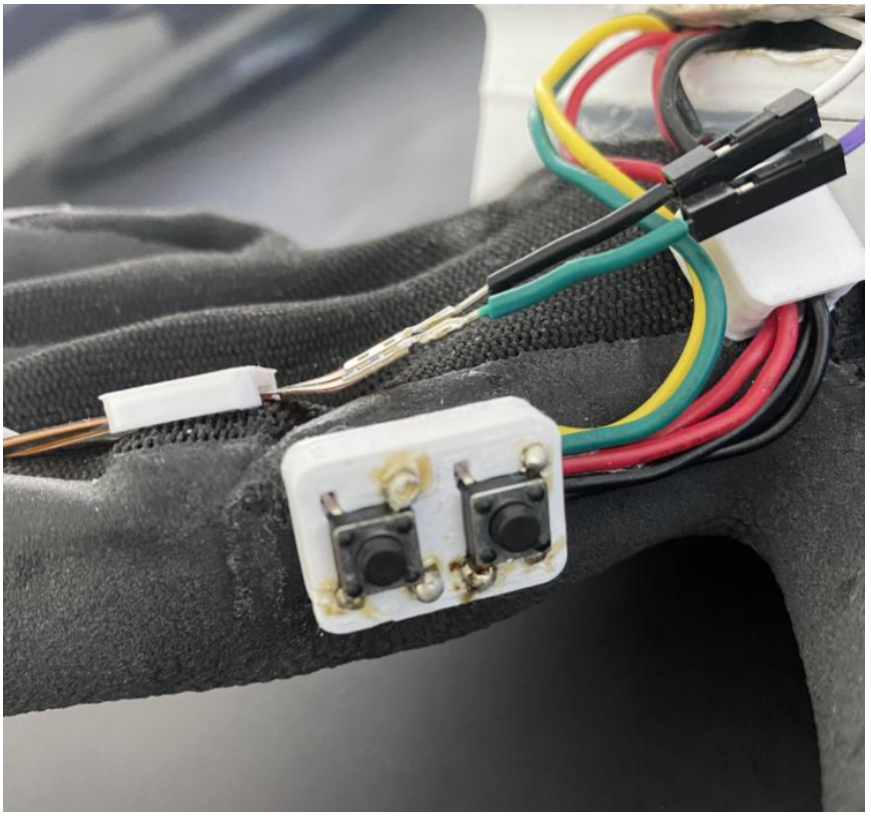
framkom olika mixar av sensorer och komponenter med olika för och nackdelar, men beslutet blev att en handske med sensorerna var det bästa valet.
Sensorerna som valts för handsken var en flexsensor och ett MPU6050 IMU. Flexsensorn behövdes bara vara fast vid toppen av pekfingret där det sedan löddes på sladdar för att kopplas till en breadboard placerad på mitten av handsken.
Breadboarden som hade MPU6050 på sig samt teensy-MCU:n och transceivern valdes att placeras på toppen av handsken. Två knappar valdes också för att kontrollera yaw på drönaren och dessa placerades på utsidan av pekfingret i en hylsa för att minska obehag av att trycka då dess kontaktpunkter kunde kännas genom handsken.
Preliminära ritningen blev enligt figur 5 nedan:



Figur 5. Preliminär ritning av handkontrollern, flexsensorn på pekfingret kontrollerar throttle. Knapparna i knapphylsan kontrollerar yaw. MPU6050n som sitter på breadboarden mäter handens lutningar och kontrollerar pitch och roll. Strömförsörjningen sker genom att koppla batterierna till teensy-MCU:n som vidare är kopplad till resterande komponenter.

4.2.2 Knapphylsa

När det kom till designen av knapphylsan behövde den vara tillräckligt stor för att hålla knapparna och ge stabilitet till dess fasthållning men inte så stor att det blir otympligt för användaren att komma åt. Vid mätningar kom en design fram enligt bilaga A. Detta printades sedan ut på en 3D- printer och fästes på handsken enligt figur 6 nedan.

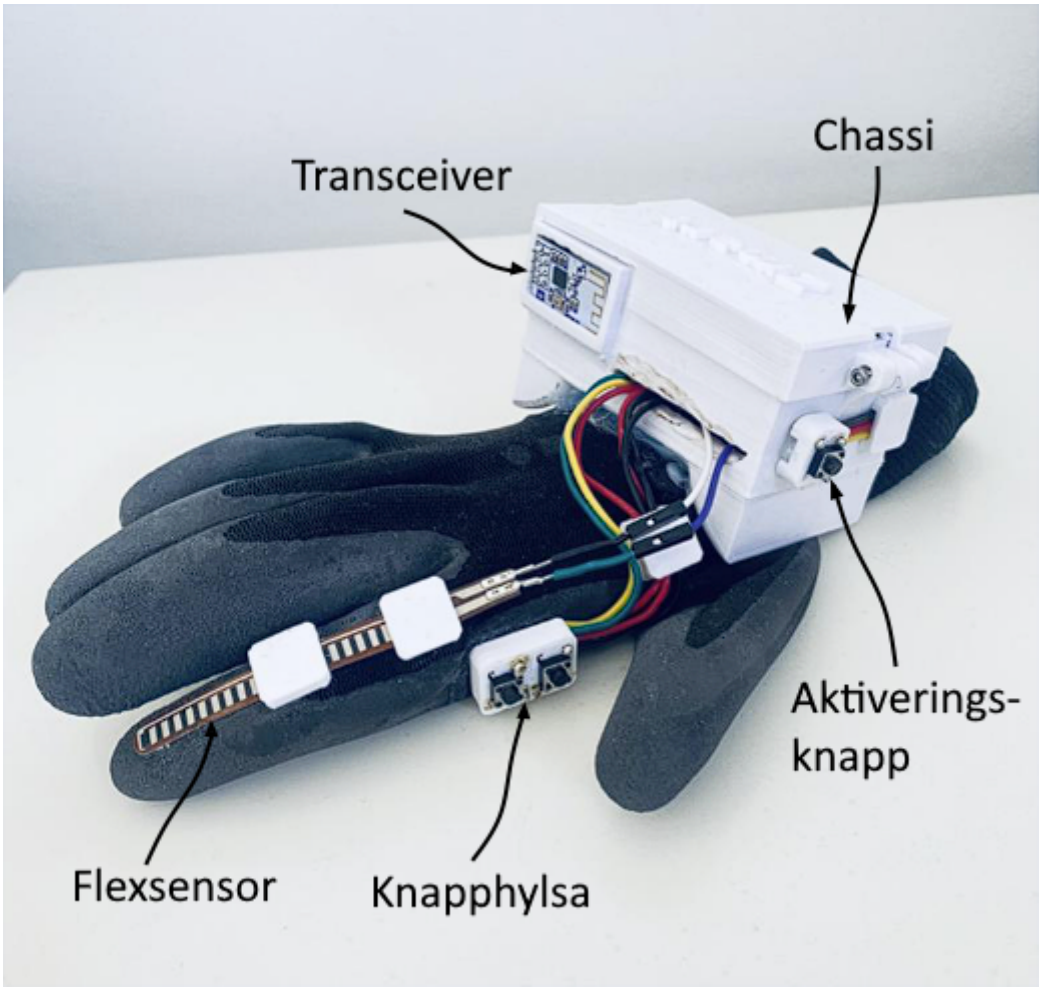


Figur 6. Utprintad knapphylsa med knappar fastlödda fäst på handsken.

4.2.3 Chassi för mikrokontroller

Det första steget i att designa ett hölje för resterande komponenter är att ta hänsyn till deras storlek och form. Breadboarden, Teensy-MCU:n och transceivern är alla relativt små och platta, så det borde vara möjligt att designa ett kompakt hölje som kan innehålla dem alla. Precis som för knapphylsan, var det bästa alternativet att cadda och 3D-printa ett hölje som passade dimensionerna av dessa komponenter. Det var också nödvändigt att kunna öppna och stänga behållaren för tillgång samt att ha öppningar för kablar. Designen blev enligt bilaga B och bilaga C. Detta hölje placerades sedan på mitten av handsken och med 3D- printade stöd limmades på enligt bilaga D. Avslutningsvis fästes batterikällan på handflatan av handsken, och lösa sladdar fästes på handsken genom en sladdhållare enligt bilaga E. Det fästes också

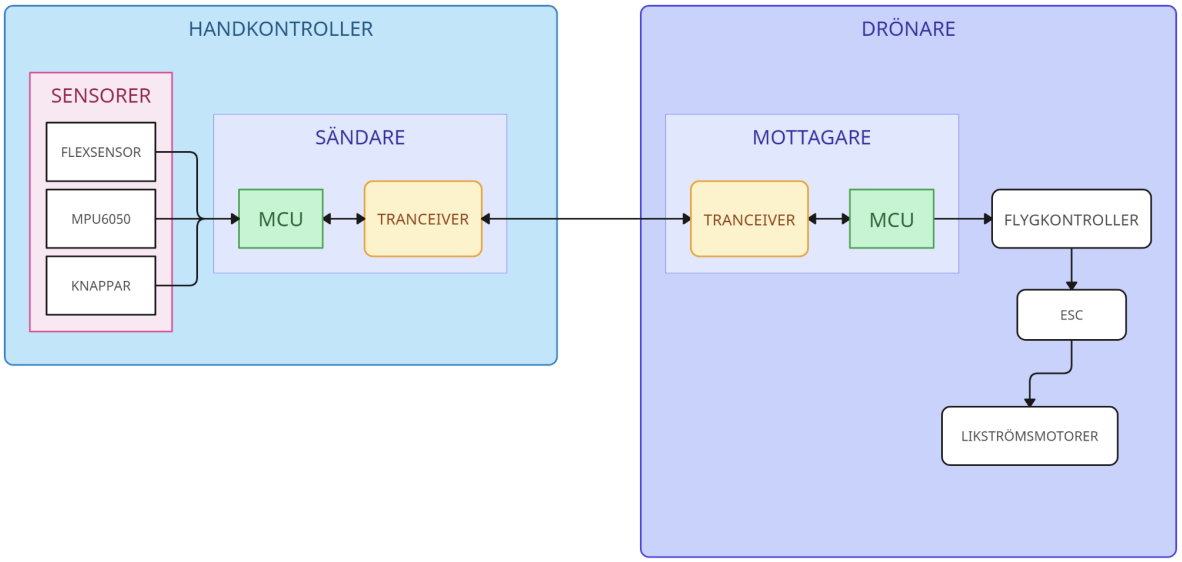
en aktiveringsknapp som kan aktivera eller avaktivera drönarens propellrar vid sidan av chassit. Den slutgiltiga handsken med alla ovan komponenter kan ses i figur 7.



Figur 7. Fullständigt utvecklad handkontroller med chassi/knapphylsa/aktiveringsknapp och komponenter fastsatta. Batteripack är fäst på undersidan.

4.2.4 Konstruktionsdesign

Konstruktionen av det nya systemet i sin helhet bygger vidare på det befintliga systemet vilket kan ses i figur 1. Skillnaden är att mottagaren i figur 1 byts ut mot en ny mottagare bestående av en MCU och en transceiver samt att sändaren enligt figuren motsvarar ett delsystem av kontrollern och består av en MCU och en transceiver. Det nya systemets uppbyggnad kan ses i figur 8.



Figur 8. Blockschema över utökade komponenter till handkontroller/drönare.

4.3 Programmering av mikrokontrollers

Nedan förklaras hur samtliga komponenter används, hur de kommunicerar med varandra och hur logiken i programkoden ser ut för att konstruera en handstyrd kontroller med sändare och en mottagare.

4.3.1 Handsken

Följande delar behandlar teorin och koden som användes för att programmera MCU:n och komponenter på handsken. För fullständig programkod se bilaga F. För kretsschema och kopplingar se bilaga H.

4.3.1.1 MPU6050

MPU6050 IMU:n använder I2C-protokollet för att kommunicera med MCU:s och andra enheter. För att MCU:n ska kunna kommunicera med MPU6050 IMU:n med I2C-protokollet används det inbyggda Arduino-biblioteket “WIRE”. Genom att starta en sändning till MPU6050:ns adress och sedan skriva adressen till ett specifikt register i MPU6050:n så kan man därefter beroende på registrets typ läsa det aktuella värdet eller skriva ett nytt värde i det aktuella registret.

Eftersom gyroskopet är väldigt känsligt för vibrationer och högfrekventa rörelser så bör det DLPF aktiveras för att få en stabilare signal på mätvärdena. Detta görs genom att skriva ett

förbestämt värde till MPU6050ns konfigurations-registret som har adressen 0x1A. Genom att skriva 0x05 till detta register så matchar bitmönstret inställningen för att aktivera ett DLPF med bandbredden 10 Hz [10].

Den råa datan för vinkelhastigheten som gyroskopet producerar är inte samma skala som verkligheten. Denna skalan går att ställa in och bör ställas in beroende på hur snabbt gyroskopet kommer att roteras och önskad precision. Registret för denna inställningen finns på adressen 0x1B och för detta projektets syfte så räcker +/- 500°/s, alltså en räckvidd på 1000°/s. Detta ställs in genom att skriva 0x08 vilket matchar det förbestämda bitmönstret för denna inställning till registret på 0x1B [10]. Den råa datan från gyroskopet för varje axel är 2 bytes eller 16 bitar. Skalan som den råa datan kommer att ha blir då:

$$\frac{2^{16}}{1000} \approx 65,5\ (°/s)^{-1}. \qquad (7)$$

Uträkningen från ekvation (7) innebär att den råa datan som läses av från gyroskopet behöver divideras med 65,5 för att få den verkliga skalan i grader per sekund.

Innan man kan behandla meningsfulla mätvärden från gyroskopet så behöver den kalibreras. Detta är viktigt eftersom den visar ett stadigt fel när den står stilla. För att kalibrera gyroskopet så läses dess mätvärden flera gånger när den är helt stilla och ett medelvärde av dessa fel beräknas. Detta medelvärdet kan sedan subtraheras från de riktiga mätningar för att ta felet i beaktning.

Registerna för den råa datan börjar vid adressen 0x3B. Där befinner sig först 6 register/bytes för accelerometers data för samtliga axlar, sedan 2 bytes för en temperatursensors data(används inte), och därefter ytterligare 6 bytes för gyroskopets data för samtliga axlar [10]. Datan för accelerometern och gyroskopet läses av och sparas i variabler för varje axel och används för att beräkna vinkeln som MPU6050 IMU:n har enligt ett complementary filter. Complementary filtret används för att kombinera både gyroskopet och accelerometers mätningar och på så sätt få en mer korrekt vinkel. För att beräkna vinkeln enligt ett complementary filter så behövs först att den råa datan bearbetas [9]. Gyroskopets data behöver vara i grader/sekund vilket uppfylls genom att dividera med den tidigare nämnda skalan: 65,5. Accelerometerdata används för att beräkna vinklarna runt x- och y-axeln. Detta görs med ekvationerna (1) och (2). Därefter sammanfogas gyroskopets skalade värde och den beräknade vinkeln från accelerometern för x- och y-axlarna enligt ett complementary filter

beskrivet i ekvationerna (3) och (4). Efter att vinklarna har beräknats genom ett complementary filter så används dessa för att beräkna vilka styrsignaler som skall skickas till quadcoptern.

4.3.1.2 Flexsensor

Eftersom flexsensorn är en resistiv sensor vars resistans beror på hur böjd sensorn är så kan man med en enkel elektrisk krets göra en spänningsdelare med hjälp av en resistor och därefter läsa av spänningen i mikrokontrollern via funktionen analogRead för att ta reda på hur böjd flexsensorn är.

4.3.1.3 Styrsignaler

De styrsignaler som behövs för att styra drönaren är: roll, pitch, yaw och throttle. För att drönaren ska stå still i luften så behöver värdena för roll, pitch och yaw vara 1500. Därför borde styrsignalerna som skickas vara 1500 när handens vinkel är 0° relativt marken och ingen knapp för yaw är intryckt. Vidare, så är det svårt för användaren att hålla handen helt stilla och plan mot marken, därför borde styrsignalerna för roll och pitch fortsätta att vara 1500 upp tills ett gränsvärde för vinkeln passeras åt vardera riktning. För att motverka att drönaren flyger för snabbt och att styrsignalerna ska stämma överens med handens vinkel så behövs ett högre gränsvärde inom intervallen -90° till 90° där drönarens styrsignal inte längre ändras och återgår till 1500 då vinkeln går över det högre gränsvärdet. Därefter så är det önskvärt att handens rörelse ger en linjärt ökande/minskande styrsignal relativt den maximala styrsignalen som skickas när handen befinner sig vid det övre gränsvärdet. Därför beror ändringen av styrsignalerna för roll och pitch på relationen av handens vinkel och det övre gränsvärdet. Därefter kan denna relationen multipliceras med en känslighetsfaktor för att bestämma styrsignalerna för roll och pitch. Styrsignalerna för roll och pitch räknas alltså ut med följande ekvation:

$$styrsignal = 1500 + r \cdot känslighetsfaktor, \quad (8)$$

där r är relationen mellan vinkeln och det övre gränsvärdet. Relationen r befinner sig inom intervallet -1 till 1 då en större eller mindre vinkeln än gränsvärdena inte ska användas. Eftersom basvärdet för roll och pitch är 1500 och relationen mellan vinkeln och det övre gränsvärdet är inom intervallet -1 till 1 så blir intervallen som dessa signaler kan befinna sig inom:

$$1500 - känslighetsfaktor \leq styrsignal \leq 1500 + känslighetsfaktor. \quad (9)$$

Samma princip gäller även för throttle, den tidigare nämnda procentuella variabeln multipliceras med en känslighetsfaktor och adderas till styrsignalens grundvärde. Styrsignalen för yaw har också 1500 som grundvärde. Men eftersom yaw ska styras med knappar så kan ett förbestämt värde sättas direkt genom att värdet adderas eller subtraheras från 1500 beroende på vilken av knapparna som är intryckta.

4.3.1.4 nRF2401L Sändare

Transceivern nRF2401L använder SPI protokollet för att kommunicera med mikrokontrollers. Utöver det inbyggda “SPI” biblioteket i Arduino så används “RF24” biblioteket för att förenkla användningen av transceivern.

Först skapas ett RF24 objekt med variabelnamnet radio. Därefter initieras detta objekt, adressen som transceivern ska skicka till anges, den utsändande effekten sätts och därefter sätts sändarläget på genom att kalla på funktionen stopListening.

Efter att styrsignalerna har beräknas och är redo att skickas så packas dessa in i en array. Genom write funktionen inom RF24 klassen så skickas denna array via radiovågor. Funktionen write returnerar true om transceivern får tillbaka en bekräftelse på att datan har tagits emot av en mottagare, annars returnerar funktionen false. Om ingen bekräftelse kommer så anropas write upp till 10 gånger innan programmet slutar försöka skicka och återgår till att läsa sensordata och beräkna nya styrsignaler.

4.3.1.5 Aktiverings funktion

För att flyga en drönare behövs signaler för roll, pitch, yaw och throttle. Men det behövs även en aktiveringssignal för att starta drönaren och aktivera likströmsmotorerna. Därför implementerades en fysisk knapp på handkontrollern som kunde växla en boolesk variabel. När denna variabel var en logisk etta så skickades en hög signal tillsammans med resten av paket till mottagaren. När variabeln var en logisk nolla så skickas istället en låg signal till mottagaren.

4.3.2 Mottagare

Den konstruerade mottagaren består av en teensy 4.0 MCU och en nRF2401L transceiver. Stysignalerna som mottagaren läser från radiovågorna skickas vidare till flygkontrollern

enligt SBUS protokollet. För fullständig programkod se bilaga F. För kretsschema och koppling se bilaga G.

4.3.2.1 nRF2401L Mottagare

Likt den sändande transceivern skapas ett RF24 objekt som initieras, adressen och effekten anges men sen sätts mottagarläget på genom att anropa funktionen startListening.
Efter att initialiseringen och inställningarna är klara så kan transceivern läsa av radiovågorna. Detta sker i programmet genom att först anropa RF24-funktionen available för att se om det finns radiovågor att läsa, om så är fallet så används funktionen read för att spara datan (styrsignalerna) i lämpliga variabler. Om ett paket togs emot så skickas en bekräftelse automatiskt tillbaka på samma adress som paketet mottogs.

Eftersom det finns en risk att transceiverna tappar kommunikationen mellan varandra skapades en extra säkerhetsåtgärd. Om transceivern inte tar emot ett paket inom en halv sekund så ska styrsignalerna gå till sitt normalläge så att drönaren blir stabil i luften, samt att throttle ska gå till sin lägsta position. Därefter ska aktiveringssignalen bli låg, så att propellrarna slutar att rotera helt och hållet. Detta resulterar i att drönaren stabiliserar sig och efter ett kort tag stänger av motorerna vilket resulterar i att den faller till marken. Detta gör att drönaren kan falla från en hög höjd utan att landa säkert, vilket inte är optimalt men är mer önskvärt än att drönaren fortsätter flyga. Efter att mottagaren själv har satt aktiveringssignalen låg så ska inga nya styrsignaler skickas till drönaren förrän mottagaren startar om (koppla från och till ström). Detta för att undvika att quadcoptern börjar flyga eller att propellrarna börjar snurra igen om kopplingen återfås och att användaren inte riskerar skada sig på drönarens propellrar då hen går för att hämta den.

4.3.2.2 SBUS

SBUS var det valda protokollet som mottagaren skickar information på till quadcoptern. För att skicka paket med SBUS till quadcoptern användes Bolder Flight Systems bibliotek “sbus” [14]. Det är möjligt att skicka paket med SBUS protokollet utan att använda detta bibliotek men programkoden blir enklare med hjälp av detta bibliotek och dess funktioner.

Först skapades ett SbusData objekt och även ett SbusTx objekt med Serial1 som argument för att ange vilka fysiska pins som ska användas för den seriella dataöverföringen. För MCU:n som används i detta projektet motsvarar “Serial1” pin nr. 1 för ta emot seriell data och pin nr.

2 för att skicka seriell data. Eftersom mikrokontrollern i detta fall ska skicka data med SBUS protokollet så är det pin nr. 2 som ska användas och kopplas till mottagare-porten på flygkontrollern. Efter att objekten skapats initierades SbusTx objektet genom att anropa funktionen Begin.

När programmet har läst nya styrsignaler med hjälp av transceivern så ska dessa styrsignaler skickas via SBUS till quadcoptern. Först läggs vardera styrsignal och aktiveringssignalen in i korresponderande kanal hos SbusData objektet. Sedan läggs SbusData objektet in i SbusTx objektet. Därefter kan funktionen Write inom SbusTx anropas för att skicka vidare datan till quadcoptern enligt SBUS protokollet via pin nr.2. Användningen av sbus-biblioteket och de inkluderande SbusData och SbusTx objekten förenklar programkoden då de redan har funktioner för att packa variablerna i 11-bitar långa kanaler enligt SBUS protokollet och skicka paketen seriellt via en pin på MCU:n.

4.4 Funktionalitets testning

För att ta reda på hur stor känslighetsfaktor som handkontrollern behöver för att beräkna styrsignaler så flögs drönaren med en konventionell kontroller och därefter analyserades flygningen med hjälp av Betaflights Blackbox Explorer. Alla styrsignaler som drönaren registrerar sparas i flygkontrollerns minne och med hjälp av Blackbox Explorer funktionen så kan grafer av styrsignaler över tiden visas, se figur 9. Efter en analys av en flygning så kom vi fram till vilka minimala/maximala styrsignaler som är rimliga för handkontrollern att producera. Dessa intervall var 1200-1800 för roll och pitch, 950-1300 för throttle och 1400-1600 för yaw.

Figur 9. Analys av en drönarflygning och styrsignaler i Betaflight Blackbox Explorer.

Innan det var dags att provflyga med handkontrollern så testades kontrollerns funktionalitet genom att kolla i BetaFlights receiver tab om handkontrollerns styrsignaler registreras av drönaren. Här märktes att samtliga signaler var för höga. Pitch, roll, yaw och throttle var högre än 1500. Basvärdet för pitch, roll och yaw justerades till 993 på handsken, vilket då motsvarade 1500 på drönaren. Basvärdet för throttle på handsken justerades till 150. Känslighetsfaktorn för roll, pitch och throttle blev 700 för att uppnå de tidigare nämnda intervallen, och för att uppnå yaw intervallet adderades eller subtraherades 100 till yaw signalen. Det är underligt att drönaren tolkar signalerna på detta sätt och ingen logisk förklaring bakom detta har hittats än. Denna justeringen av basvärden visar sig dock vara stabil och linjär ökning/minskning av signalerna som skickas resulterar i en linjär ökning/minskning av styrsignalen som drönaren tolkar.

Efter att alla signaler såg ut att stämma i betaflight så var det dags att provflyga. Första fysiska provflygningen gick bra, alla funktioner fungerade som tänkt. Roll, pitch, yaw, throttle och aktiveringssignalen fungerade bra. Dessutom testades säkerhets funktionen genom att batteriet till handkontrollern kopplades ur, så att mottagaren tappade kommunikationen med handkontrollern och drönaren landade automatiskt som tänkt.

5. Resultat

Syftet med projektet var att konstruera en väl fungerande handkontroll som skulle kunna effektivt styra en drönare. En avgörande faktor för att uppnå detta var att samla in information på ett tillförlitligt sätt från kontrollen och säkerställa en smidig kommunikation mellan kontrollen och drönaren. Genom att genomföra omfattande funktionalitetstester under projektets gång lyckades vi nå dessa mål och resultatet blev en fullt fungerande handkontroll.

Både ZMart och vår projektgrupp är mycket nöjda med utfallet av detta projekt och ser stor potential för framtida utveckling inom de områden som projektet berörde. Vi betraktar detta som en betydelsefull framgång som öppnar upp för ytterligare innovation och förbättringar.

Detta projekt innebar ett betydande framsteg inom utvecklingen av handkontroller för drönare och har bidragit till vår förståelse av de tekniska utmaningar som är involverade i att skapa en effektiv styrningsenhet. Genom att bygga en fungerande handkontroll har vi inte bara uppnått våra ursprungliga mål utan också öppnat upp för nya möjligheter och framtida forskning. Sammanfattningsvis kan vi konstatera att projektet har varit en framgång och har uppfyllt sitt övergripande syfte.

6. Slutsats och Diskussion

I detta kapitel diskuteras projektets resultat samt om frågeställningarna i avsnitt 1.4 har blivit besvarade.

6.1 Besvarade frågeställningar

- Vilken av de trådlösa kommunikationsmetoderna: FSK, GFSK, FHSS och DSSS är mest lämplig för att styra en drönare med avseende på:
    - säkerhet?
    - räckvidd?
    - snabbhet?

Under projektets gång så uppstod flera idéer om hur kommunikationen ska hanteras mellan kontrollern och quadcoptern, alla med sina för och nackdelar. Det vi valde var att använda frekvensutrymmet i 2.4Ghz som ingår i det licensfria ISM-bandet. Detta eftersom det är en populär frekvens för civila produkter och det finns många alternativ i form av transceivers att välja på. Litteraturstudien påvisade att FHSS är den modulationsteknik som en drönare bör använda tack vare dess förmåga att motverka störningar, undvika avlyssning och dess snabbhet. Räckvidden är kopplad till hur hög utsänd effekt sändaren skickar signalerna med och hur känslig mottagaren som används är, vilket beror på modulationsteknikens känslighet mot störningar. På grund av en missuppfattning av att DSSS har en kortare räckvidd sållades den bort och FHSS valdes som den mest lämpliga kommunikationsmetoden. Men DSSS skulle också passa bra för detta projekt.

På grund av brist på moduler som använder FHSS modulering som har en tillräckligt hög utsänd effekt och därmed räckvidd, valdes istället en modul som använder GFSK modulering. Inom säkerhet är detta inte optimalt då dels störningar som befinner sig på de använda frekvenserna inte går att undvika samt att paketen som skickas inte är krypterade, vilket gör det möjligt för avlyssning och potentiellt att en tredje part kan få tag i adressen som används. Om detta sker så kan det leda till allvarliga konsekvenser då det går att konstruera en sändare som använder samma adress och därmed ta över quadcoptern. Detta kan i sin tur leda till stöld av quadcoptern eller att användaren tappar kontrollen över quadcoptern och att den kraschar. En möjlig åtgärd mot denna brist skulle kunna vara att implementera

frekvenshoppande i programkoden men valet var att inte lägga tid på detta då huvudintresset var att få en fungerande intuitiv styrning.

- Hur bör styrsignalerna som skickas till drönaren se ut?
    - Vilken information bör skickas?
    - Vilket format bör användas?

En stor fråga om styrning var exakt vad som ska skickas till drönaren för att få den att agera på ett korrekt sätt. De grundläggande signalerna som drönarens flygkort kräver för att kunna flyga är börvärden för roll, pitch, yaw och throttle. Värdena för roll och pitch är representerade så att 1500 är mittpunkten och motsvarar noll graders lutning. Högre värden motsvarar större positiv rotation och mindre värden motsvarar större negativ rotation. Även yaw har en mittpunkt på 1500 och får en konstant positiv eller negativ vinkelhastighet om en större eller mindre signal används. Värdet för throttle har ingen mittpunkt som de övriga signalerna. För throttle så innebär ett högre värde högre throttle. Vidare så får flygkontrollern dessa värden från mottagaren på drönaren. För att flygkontrollern ska kunna kommunicera med mottagaren så används ett protokoll. I projektet valdes SBUS som RX-protokoll eftersom SBUS har några fördelar gentemot andra möjliga protokoll. Generellt så stödjer de flesta flygkontrollers på marknaden flera olika protokoll och SBUS är ett av de populäraste protokollen som används inom drönar-branschen. Genom att välja SBUS som protokoll för mottagaren så ökar kompabiliteten för mottagaren och kan kombineras med andra flygkontrollers och drönare. En annan fördel som SBUS har är att det är ett seriellt protokoll. Detta innebär att det endast krävs en kabel för att kommunicera med flygkortet. Det är en klar fördel jämfört mot att använda PWM signaler vilket använder en kabel för varje kanal som används.

- Vilka sensorer bör användas för att mäta handrörelserna med avseende på:
    - noggrannhet?
    - kostnad?

I förstudien kom det fram olika lösningar på hur man skulle mäta handrörelserna. Det fanns alternativ om att mäta dem med kamera, fysiskt med flexsensorer och/eller potentiometrar men beslutet gick till att mäta alla dessa med ett gyroskop. Gyroskop enbart hade haft problem med att drifta och ge felaktiga signaler efter tid så det var viktigt att hitta ett kort där både gyroskop samt accelerometer var inbyggda då accelerometern motverkar drift från gyroskopet. Detta val gjordes då det var enklare att tillverka, billigare och mer noggrant än de

andra alternativen för samma budget. För att bestämma throttle kunde detta gjorts med en flexsensor, knappar eller en spak. Vi tyckte dock att det var mer funktionellt att böja fingret för att styra denna funktion och landade i att använda en flexsensor.

- Har den designade kontrollern samma funktionalitet som den konventionella kontrollern?

Den designade kontrollern har sina för och nackdelar. Den har mindre möjligheter till att programmera in sekundära funktioner så som att använda tillägg på drönaren vilket går att programmera in på spakarna av den konventionella kontrollern. Något som den designade kontrollern har till sin fördel är att den är mindre känslig och reagerar mindre på förändringar gentemot den konventionella. Detta kan göra den lättare att hantera för nybörjare då det minskar chansen för att drönaren flyger för snabbt utan att det var meningen.
Något som är sämre med den designade kontrollern är att det är svårt att stå stilla på plats då den styrs av hand/finger rörelser vilket kan skifta om man tappar fokus.

6.2 Utvecklingsmöjligheter

Några utvecklingsmöjligheter för detta projekt är att lägga till ökad funktionalitet så att drönaren kan utföra mer än att bara flyga. Några funktioner som kan läggas till är att lägga till fler knappar och spakar så det finns möjlighet till att använda sig av kamera eller andra tillbehör för drönaren. Dessa tillsammans med en display kan drastiskt minska komplexiteten för att finjustera drönarens funktionalitet utan att behöva koda in det i MCU:n.
En sista förbättring för att utveckla vidare är att använda sig av en bindnings funktion som ökar möjligheterna till att koppla handkontrollern till andra mottagare eller koppla om utan att orsaka problem.

Referenser

[1] K. Daware, “HOW A DC MOTOR WORKS,” *Electrical easy*, [Online] u.d. Tillgänglig: https://www.electricaleasy.com/2014/01/basic-working-of-dc-motor.html (hämtad: 2023-06-05).

[2] S. Vorkoetter, “An Electronic Speed Control Primer,” *Stefanv*, [Online], Sep. 30, 1997. Tillgänglig: http://www.stefanv.com/electronics/escprimer.html (hämtad: 2023-06-04)

[3] A. Sigalos, M. Papoutsidakis, A. Chatzopoulos, D. Piromalis, “DESIGN OF A FLIGHT CONTROLLER AND PERIPHERALS FOR A QUADCOPTER,” *International Journal of Engineering Applied Sciences and Technology*, vol. 4, nr. 5, ss. 463-470, Sep, 2019, doi: 10.33564/ijeast.2019.v04i05.067.

[4] RF Wireless World, “Transmitter vs Receiver-difference between transmitter and receiver types,” u.d. [Online], Tillgänglig: https://www.rfwireless-world.com/Terminology/difference-between-transmitter-and-receiver-types.html

[5] Pk0001, “Quadrotor-xyz,” (2019) [Elektronisk bild]. Tillgänglig: https://commons.wikimedia.org/wiki/File:Quadrotor-xyz.svg (hämtad 2022-05-09)

[6] J. Greg, R. Scott, K. Eunhye, “Engaging children in engineering design through the world of quadcopters,” *Children’s technology and engineering*, vol. 21, ss. 7-11, Maj. 2019. [Online]. Tillgänglig: https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=6484&context=facpub, Hämtad: 2023-06-04.

[7] Betaflight, “Receiver Tab”, 2023 [Online], Tillgänglig: https://betaflight.com/docs/wiki/configurator/receiver-tab (Hämtad 2023-05-09)

[8] G. Zhanshe et al., “Research development of silicon MEMS gyroscopes: a review,” *Microsystems Technology*, vol. 21, ss. 2053-2066, Aug. 2015. doi: 10.1007/s00542-015-2645-x.

[9] P. Gui, L. Tang, S.Mukhobadhyay, “MEMS Based IMU for Tilting Measurement” i *Conf. on Industrial Electronics and Applications (ICIEA)*, 2015, ss. 2007-2009, [Online]. Tillgänglig: https://www.researchgate.net/profile/Pengfei_Gui/publication/308850497_MEMS_based_IMU_for_tilting_measurement_Comparison_of_complementary_and_kalman_filter_based_data_fusion/links/5a7832450f7e9b41dbd26dfe/MEMS-based-IMU-for-tilting-measurement-Comparison-of-complementary-and-kalman-filter-based-data-fusion.pdf, Hämtad 2023-06-03.

[10] *MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2*, San Jose, USA: IvenSense, 2013. [Online], Tillgänglig: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf, Hämtad: 2023-05-09.

[11] *Flex Sensor*, Salt Lake City, USA: Spectra Symbol, u.d [Online]. Tillgänglig: https://www.electrokit.com/uploads/productfile/41010/flex22.pdf, Hämtad: 2023-06-05.

[12] Elprocus, “RF Module-Transmitter & Receiver,” 2023. [Online]. Tillgänglig: https://www.elprocus.com/rf-module-transmitter-receiver/ (hämtad: 2023-06-05)

[13] *nRF2401L Single Chip 2.4 GHz Transceiver*, Trondheim, Norge: Nordic Semiconductor, 2007. [Online]. Tillgänglig: https://www.mouser.com/datasheet/2/297/nRF24L01_Product_Specification_v2_0-9199.pdf, Hämtad: 2023-06-05.

[14] Boulder Flight Systems, “sbus,” GitHub, [Online], Nov. 14, 2022. Tillgänglig: https://github.com/bolderflight/sbus (hämtad: 2023-06-05).

[15] K. Hemmanur, “Inter-Integrated Circuit (I2C),” Michigan State University, East Lansing, USA, 2009. [Online]. Tillgänglig:

https://www.egr.msu.edu/classes/ece480/capstone/fall09/group03/AN_hemmanur.pdf, Hämtad: 2023-06-05.

[16] Wootton. C, "Serial Peripheral Interface (SPI)," i *Samsung ARTIK Reference, Red*. Berkeley, USA: Apress, 2016, ss. 335-349. [Online]. Tillgänglig: https://link.springer.com/chapter/10.1007/978-1-4842-2322-2_21, Hämtad: 2023-06-05.

[17] M. K. Parai, B. Das, G. Das, "An Overview of Microcontroller Unit: From Proper Selection to Specific Application," *International Journal of Soft Computing and Engineering (IJSCE)*, Vol. 2, nr. 6, ss. 2231-2307, Jan. 2013. [Online]. Tillgänglig: https://www.ijsce.org/wp-content/uploads/papers/v2i6/F1161112612.pdf, Hämtad: 2023-06-04.

[18] PJRC, "Teensy(R) 4.0 Development Board," u.d. [Online]. Tillgänglig: https://www.pjrc.com/store/teensy40.html (hämtad: 2023-06-05).

[19] D. Norris, *Build Your Own Quadcopter: Power Up Your Designs with the Parallax Elev-8*. 1. uppl., McGraw-Hill Education, 2014. [Online]. Tillgänglig: https://www.accessengineeringlibrary.com/content/book/9780071822282/chapter/chapter6, Hämtad: 2023-06-05.

[20] B. Watson, "FSK: Signals and Demodulation," *Watkins-Johnson Tech-note*, vol. 7, nr. 5, Okt. 1980, Tillgänglig: https://web.archive.org/web/20120907031840/http://www.xn--sten-cpa.se/share/text/tektext/digital-modulation/FSK_signals_demod.pdf

[21] everythingRF, "What is GFSK Modulation?," 2022. [Online]. Tillgänglig: https://www.everythingrf.com/community/what-is-gfsk-modulation (hämtad: 2023-06-05)

[22] N. H. Motlagh, "Frequency Hopping Spread Spectrum: An Effective Way to Improve Wireless Communication Performance," Department of Information Technology, Vaasa University of Applied Sciences, Finland, [Online]. Tillgänglig: https://www.researchgate.net/profile/Naser-Hossein-Motlagh/publication/221905791_Freque

ncy_Hopping_Spread_Spectrum_An_Effective_Way_to_Improve_Wireless_Communication_Performance/links/5405b55f0cf2bba34c1d7a75/Frequency-Hopping-Spread-Spectrum-An-Effective-Way-to-Improve-Wireless-Communication-Performance.pdf, Hämtad: 2023-06-05

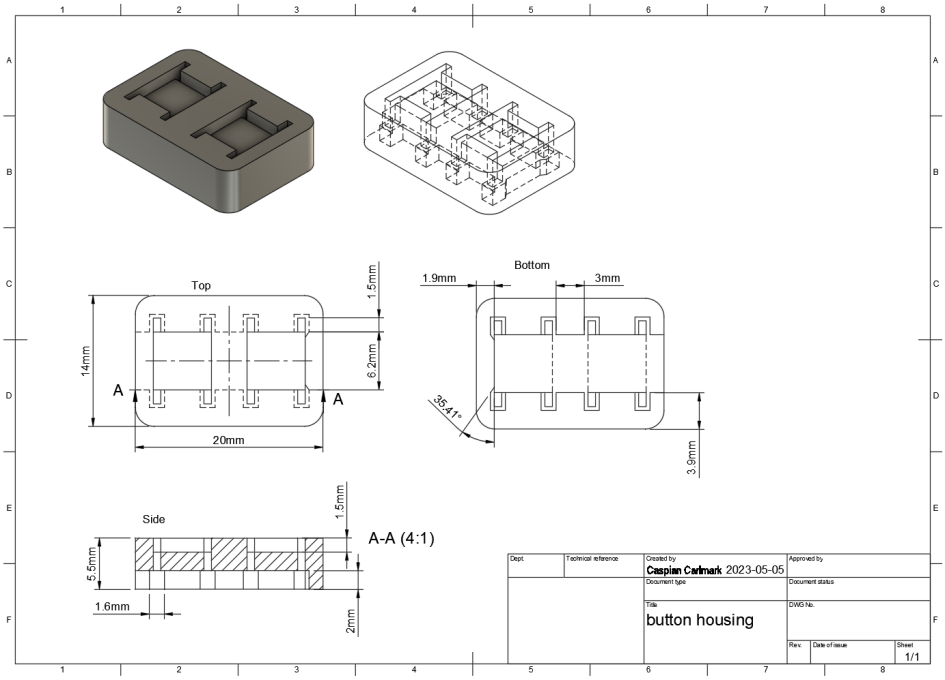
[23] M. Hasan, J. M. Thakur, P. Podder, "Design and Implementation of FHSS and DSSS for Secure Data Transmission," *International Journal of Signal Proccessing Systems*, Vol. 4, No. 2, ss. 144-149, Jan. 2015, doi:10.12720/ijsps.4.2.144-149

[24] A. Bensky, "Radio propagation," i *Short- range Wireless Communication*: 3. uppl., Amsterdam, Nederländerna: Elsevier, 2019, kap. 2, ss. 11-41. Tillgänglig: https://www.sciencedirect.com/science/article/pii/B9780128154052000026, Hämtad: 2023-06-05.

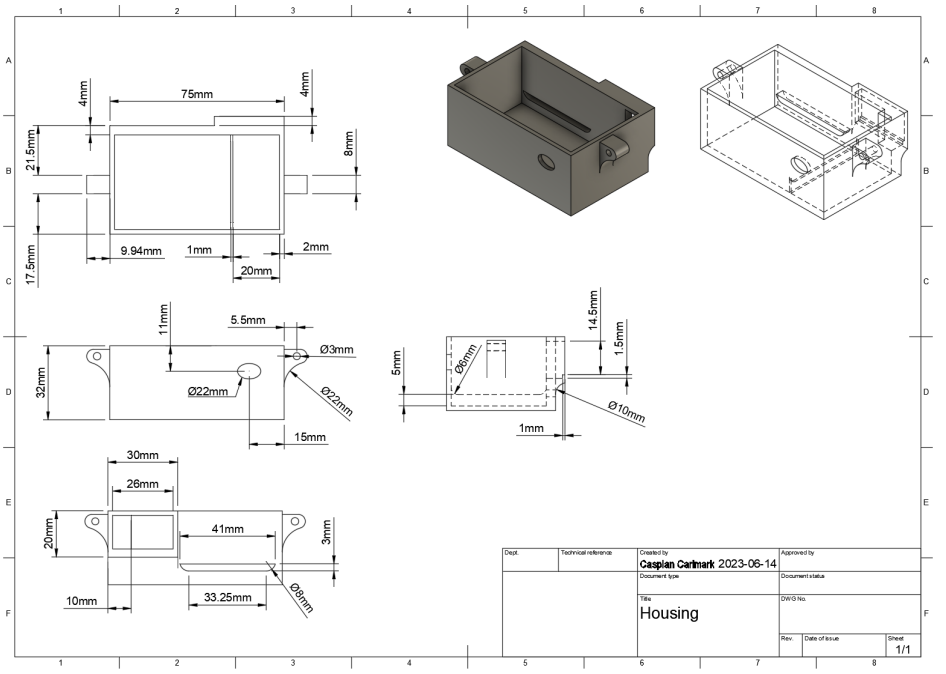
[25] *Post- och telestyrelsen föreskrifter om undantag från tillståndsplikt för användning av vissa radiosändare*, Post- och telestyrelsen, Stockholm, Sverige, Dec, 2014. [Online]. Tillgänglig: https://pts.se/globalassets/startpage/dokument/legala-dokument/foreskrifter/radio/ptsfs-2014_5-undantag-tillstandsplikt-rattad.pdf

Bilagor

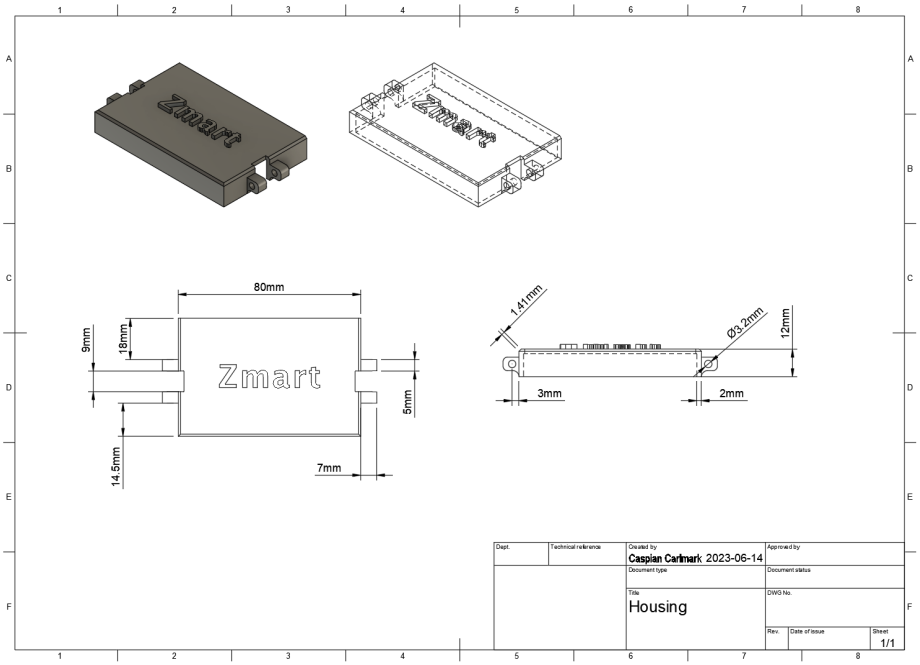
Bilaga A - Knapp Hylsa



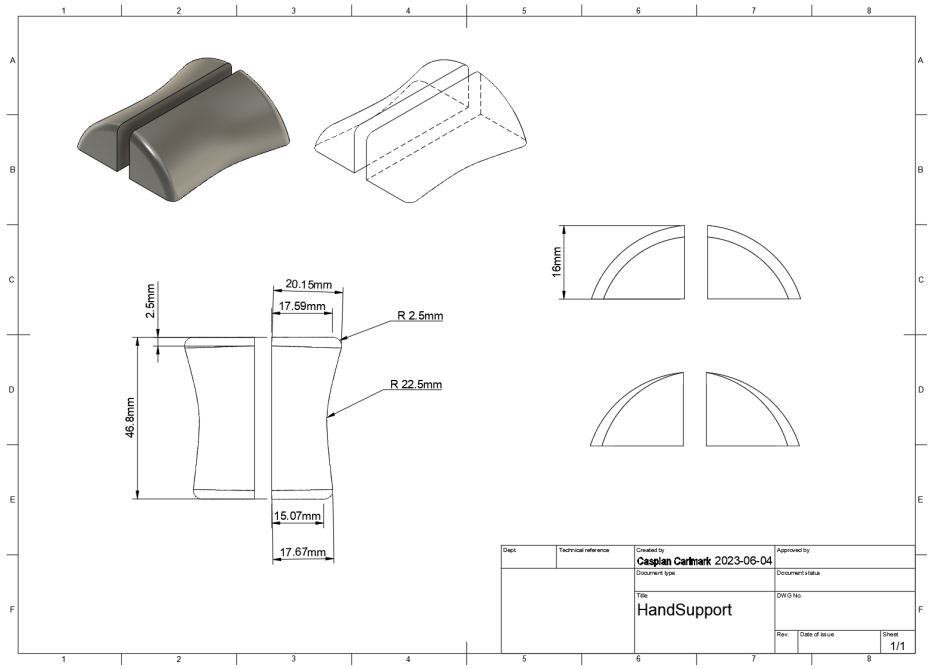
Bilaga B - Hållare



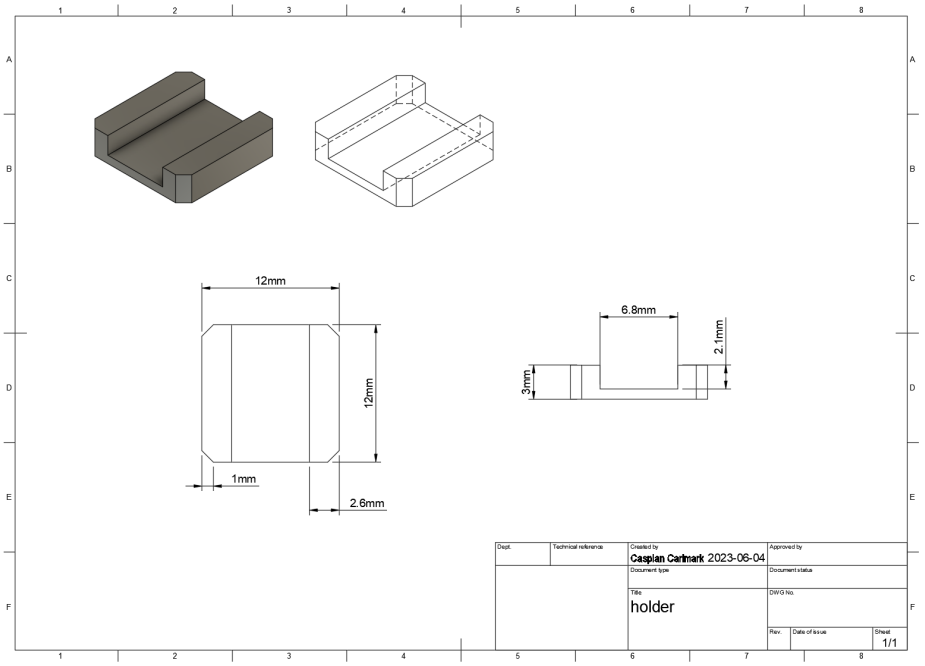
Bilaga C - Hållare lock



Bilaga D - Stöd för hållare



Bilaga E - Sladdhållare



Bilaga F - Kod för handkontroller

```
#include <Wire.h> //i2c protocoll för MPU6050
#include <RF24.h> //radio
#include <SPI.h>  //serial peripheral interface, behövs för RF24(nRF24L01)

//Här defineras addresser och pins som används
#define MPU6050address 0x68
#define flexUpp 14
#define flexNer 15
#define CE 7
#define CSN 8
#define yawLeft 0
#define yawRight 1
#define armPin 3

//Variabler för MPU6050
int16_t GYRO_XOUT, GYRO_YOUT, GYRO_ZOUT, ACCEL_XOUT, ACCEL_YOUT,
ACCEL_ZOUT;  //raw data från MPU6050
float accXAngle, accYAngle, accZAngle; //vinklar från accel
float gyroXDegps, gyroYDegps, gyroZDegps, gyroXOffset = 0, gyroYOffset = 0,
gyroZOffset = 0;  //raw gyrodata till grader/s samt offset
float gyroXAngle = 0, gyroYAngle = 0, gyroZAngle = 0; //vinkel från gyro, integrerat
gyroDegps
float angleX = 0, angleY = 0, angleZ = 0; //vinklar från complementary filter
float dt; //används för tid mellan mätningar gyro
float radiansToPi = 180/PI;
unsigned long lasttime; //tid vid senast mätning
float alpha = 0.7;  //konstant för filter, alpha = tau/(tau+dt), tau tidskonstant

//variabler för flex
int valUpp; //lästa värdet för flexUpp sensorn
float cutOffLow = 190; //DESSA ÄR GODTYCKLIGA ATM, värdet för när throttle börjar
reagera
float cutOffHigh = 110; //värde för när throttle ska vara max
float percentUpp; //percent för hur mycket flexsensor är böjd

//variabler för nRF24L01 och styrsignaler
int minAngle = 10; // om vinkel är under 15 grader gör ingeting
int maxAngle = 70; // om vinkel är över 70, gör inget, (vinkel hoppar runt efter 90 grader pga
atan)
float sensitivity = 700.0;
int yawSensitivity = 100;
int roll, pitch, yaw, throttle;
const byte radioAddress[6] = "ZMART"; //radioaddress
RF24 radio(CE, CSN);  //skapa RF24 objekt
int payload[5];  //en array med alla styrsignaler som ska skickas

bool armed = false;
bool prevArm = false;
```

```
bool currArm = false;

/*
Konfigurerar och startar MPU6050.
Startar lowpass filter för gyro, 10 Hz bandwidth.
Ställer in scale mätvärden för gyro.
Beräknar gyro offset och sparar i globala variabler: gyro(X/Y/Z)Offset.
*/
void setupMPU6050(){
  Wire.begin();
  Wire.beginTransmission(MPU6050address);
  Wire.write(0x1A); //Config register
  Wire.write(0x05); // 10 Hz bandwidth DLPF (digital low pass filter)
  Wire.endTransmission();
  Wire.beginTransmission(MPU6050address);
  Wire.write(0x1B); //Gyroscope configuration register
  Wire.write(0x08); //Range och scale mätvärden av gyro (+/- 500 deg/s och 65,5 lsb per
deg/s)
  Wire.endTransmission();
  Wire.beginTransmission(MPU6050address);
  Wire.write(0x6B); //Power management 1 register
  Wire.write(0x00); //Nollställ för att starta
  Wire.endTransmission();

  //Beräkna offset för gyro genom att ta medelvärde av mätningar när den är stilla.
  int iterations = 1000;
  int i = 0;
  float gyroXOffset_tmp = 0, gyroYOffset_tmp = 0, gyroZOffset_tmp = 0;
  while (i<iterations){
    readMPU6050();
    gyroXOffset_tmp += gyroXDegps;
    gyroYOffset_tmp += gyroYDegps;
    gyroZOffset_tmp += gyroZDegps;
    i++;
    delay(1);
  }
  gyroXOffset = gyroXOffset_tmp/iterations; //Offseten är medelvärdet
  gyroYOffset = gyroYOffset_tmp/iterations;
  gyroZOffset = gyroZOffset_tmp/iterations;
}


/*
Läser av data från sensorerna.
Omvandlar till rikriga värden.
Kallar på calculateAngles, så värden uppdateras efter rop på denna funktionen
Sparar värden i globala variabler: ACCEL_(X/Y/Z)OUT, GRYO_(X/Y/Z)OUT,
gyro(X/Y/Z)Degps
*/
void readMPU6050(){
```

```
  Wire.beginTransmission(MPU6050address);
  Wire.write(0x3B); //Första register för märvärden
  Wire.endTransmission(false);  //Håll kvar connection
  Wire.requestFrom(MPU6050address, 14); //Hämta 14 bytes. (6 accel, 2 temp, 6 gyro)
  ACCEL_XOUT = Wire.read()<<8 | Wire.read();
  ACCEL_YOUT = Wire.read()<<8 | Wire.read();
  ACCEL_ZOUT = Wire.read()<<8 | Wire.read();
  Wire.read();  // Gå förbi tempvärden
  Wire.read();
  GYRO_XOUT = Wire.read()<<8 | Wire.read();
  GYRO_YOUT = Wire.read()<<8 | Wire.read();
  GYRO_ZOUT = Wire.read()<<8 | Wire.read();

  gyroXDegps = (GYRO_XOUT/65.5)-gyroXOffset;  //Gör om rå sensordata till verklig
storlek, justera efter offset.
  gyroYDegps = (GYRO_YOUT/65.5)-gyroYOffset;
  gyroZDegps = (GYRO_ZOUT/65.5)-gyroZOffset;

  calculateangles();
}

/*
Använder sensordata för att beräkna vinklar.
Använder complementary filter för att slå ihop gyro och accel för att få en mer accurate
vinkel.
Sparar värden i globala variabler: Gyro(X/Y)Angle, acc(X/Y)Angle, angle(X/Y).
*/
void calculateangles(){
  dt=((float)micros()-(float)lasttime)/1000000; //beräkna tid sedan senaste mätningen
  lasttime=micros();  //senaste mätningen är nu

  gyroXAngle = gyroXAngle + gyroXDegps*dt;  //Vinkel beräknat bara av gyro, används inte
  gyroYAngle = gyroYAngle + gyroYDegps*dt;

  accXAngle = atan((float)ACCEL_YOUT/(float)ACCEL_ZOUT)*radiansToPi;  //Vinkel
beräknat av accelerometern
  accYAngle = -atan((float)ACCEL_XOUT/(float)ACCEL_ZOUT)*radiansToPi;

  //Complementary filter, enligt: MEMS Based IMU for Tilting Measurement
  if(isnan(angleX)||isnan(angleY)){ //om nan, nollställ för att börja beräkningar.
    angleX = 0;
    angleY = 0;
  }
  angleX = alpha*(angleX+gyroXDegps*dt)+(1-alpha)*accXAngle;
  angleY = alpha*(angleY+gyroYDegps*dt)+(1-alpha)*accYAngle;
  Serial.print("angleX is :");
  Serial.println(angleX);
  Serial.print("angleY is :");
  Serial.println(angleY);
}
```

```
/*
Funktion för att printa ut vinklar och gyrodata till Serial Monitor
*/
void printAngles(){
  Serial.print("X: ");
  Serial.print(angleX);
  Serial.print("    ");
  Serial.print("Y: ");
  Serial.println(angleY);

  Serial.print("accelXAngle: ");
  Serial.print(accXAngle);
  Serial.print("    ");
  Serial.print("accelYAngle:" );
  Serial.println(accYAngle);
  Serial.print("accelZ:" );
  Serial.println(ACCEL_ZOUT);

  Serial.print("gyroXDegps: ");
  Serial.print(gyroXDegps);
  Serial.print("    ");
  Serial.print("gyroYDegps:" );
  Serial.println(gyroYDegps);

  Serial.print("dt: ");
  Serial.println(dt);

  Serial.print("alpha: ");
  Serial.println(alpha);
}

/*
Läser av flexsensorer och beräknar procent av hur throttle bör ändras
(0% till +100%).
Sparar värden i global variabel: percentUpp.
*/
void readFlexSensors(){
  valUpp = analogRead(flexUpp);
  if (valUpp < cutOffLow){  //om värdet mindre än cutoff
    percentUpp = 100-100*((valUpp-cutOffHigh)/(cutOffLow-cutOffHigh));  //hur många
procent är sensorn mellan cutOffHigh och cutOffLow
    if(percentUpp > 100){
      percentUpp = 100;
    }
  }
  else{ //annars ändra inte throttle
    percentUpp = 0;
  }
}
```

```
/*
Printa ut värdet som läses från flexSensorer och de beräknade procenten.
*/
void printFlex(){
  Serial.print("valUpp: ");
  Serial.print(valUpp);
  Serial.print("    ");
  Serial.print("percentUpp: ");
  Serial.println(percentUpp);
}

/*
Starta radio, ge address, sätt låg power för att skippa buggar
*/
void setupRadio(){
  radio.begin();
  radio.openWritingPipe(radioAddress);
  radio.setPALevel(RF24_PA_LOW);
  radio.stopListening();
}

/*
Skickar payloaden genom transciever(sändare) till transciever(mottagare).
Serial prints för debugging
*/
void radioSend(){
  for(int i = 0; i<10; i++){
    if (radio.write(payload, sizeof(payload))){
      //Serial.print("payload sent after ");
      //Serial.print(i);
      //Serial.println("Failed tries");
      return;
    }
    delay(10);
  }
  //Serial.println("Failed send payload after 10 tries");
}

/*
Beräknar payload(styrsignalerna) utifrån sensorvärden och gränser
993 motsvarar 1500 på drönaren...
*/
void calculatePayload(){
  if(((int)angleX<minAngle && (int)angleX>(-1*minAngle)) || ((int)angleX<(-1*maxAngle))
|| (int)angleX>maxAngle || ACCEL_ZOUT<0){  //om -minAngle>angleX>minAngle, eller
över maxAngle/under -maXangle stå stilla, accel_zout för det inte ska gå uppåner
    //roll = 1500;
    roll = 993;
```

```
  }
  else{
    //roll = 1500 + (angleX/(float)maxAngle)*sensitivity; //Annars, procent av angleX mot
maxAngle*sensitivity
    roll = 993 + (angleX/(float)maxAngle)*sensitivity; //Annars, procent av angleX mot
maxAngle*sensitivity
  }

  if(((int)angleY<minAngle && (int)angleY>(-1*minAngle)) || (int)angleY<(-1*maxAngle) ||
(int)angleX>maxAngle || ACCEL_ZOUT<0){  //samma för pitch fast med angleY
    //pitch = 1500;
    pitch = 993;
  }
  else{
    //pitch = 1500 + (angleY/(float)maxAngle)*sensitivity;
    pitch = 993 + (angleY/(float)maxAngle)*sensitivity;
  }

  yaw = 993;
  if(digitalRead(yawLeft)){ //om knapp för yaw vänster är intryckt...
    yaw += yawSensitivity;
  }
  if(digitalRead(yawRight)){  //om knapp för yaw högner är intryckt...
    yaw -= yawSensitivity;
  }
  throttle = 150 + sensitivity*percentUpp/100;  //throttle är sitt basvärde och +
procent*senisitivity.

  payload[0]=roll;    //lägg till signalerna i paketet som ska skickas
  payload[1]=pitch;
  payload[2]=yaw;
  payload[3]=throttle;

  prevArm = currArm;    //följande för att toggla arm vid knapptryck.
  delay(5);
  currArm = digitalRead(armPin);
  delay(5);
/*
  Serial.print("prevArm: ");
  Serial.println(prevArm);
  Serial.print("currArm: ");
  Serial.println(currArm);
  */
  if(!prevArm && currArm){
    armed = !armed;
    //Serial.print("Armed: ");
    //Serial.println(armed);
  }

  if(armed){    //om armad, skicka hög signal, annars låg.
```

```
    payload[4]=1800;
  }
  else{
    payload[4]=500;
  }
}

//hjälpfunktion för att printa innehåll i payload
void printRPYT(){
  Serial.print("Roll: ");
  Serial.println(payload[0]);
  Serial.print("Pitch: ");
  Serial.println(payload[1]);
  Serial.print("Yaw: ");
  Serial.println(payload[2]);
  Serial.print("Throttle: ");
  Serial.println(payload[3]);
}

void setup() {
  delay(300);
  Serial.begin(9600);
  while(gyroXOffset == 0 && gyroYOffset == 0 && gyroZOffset == 0){  //försök kalibrera
på nytt sålänge kalibreringen inte fungerar.
    setupMPU6050();
  }
  Serial.println("Gyro calibration completed");
  Serial.print("GyroXOffset: ");
  Serial.println(gyroXOffset);
  Serial.print("GyroYOffset: ");
  Serial.println(gyroYOffset);
  Serial.print("GyroZOffset: ");
  Serial.println(gyroZOffset);

  pinMode(flexUpp, INPUT);
  pinMode(flexNer, INPUT);
  pinMode(yawLeft, INPUT);
  pinMode(yawRight, INPUT);
  pinMode(3, INPUT);
  setupRadio();
}


void loop() {
  //MPU6050, läs av för att uppdatera vinklar
  readMPU6050();
  printAngles();
  //flexsensor
  readFlexSensors();
  //printFlex();
```

```
  //payload uträkning
  calculatePayload();
  //printRPYT();
  //nRF24L01
  radioSend();
  delay(50);
}
```

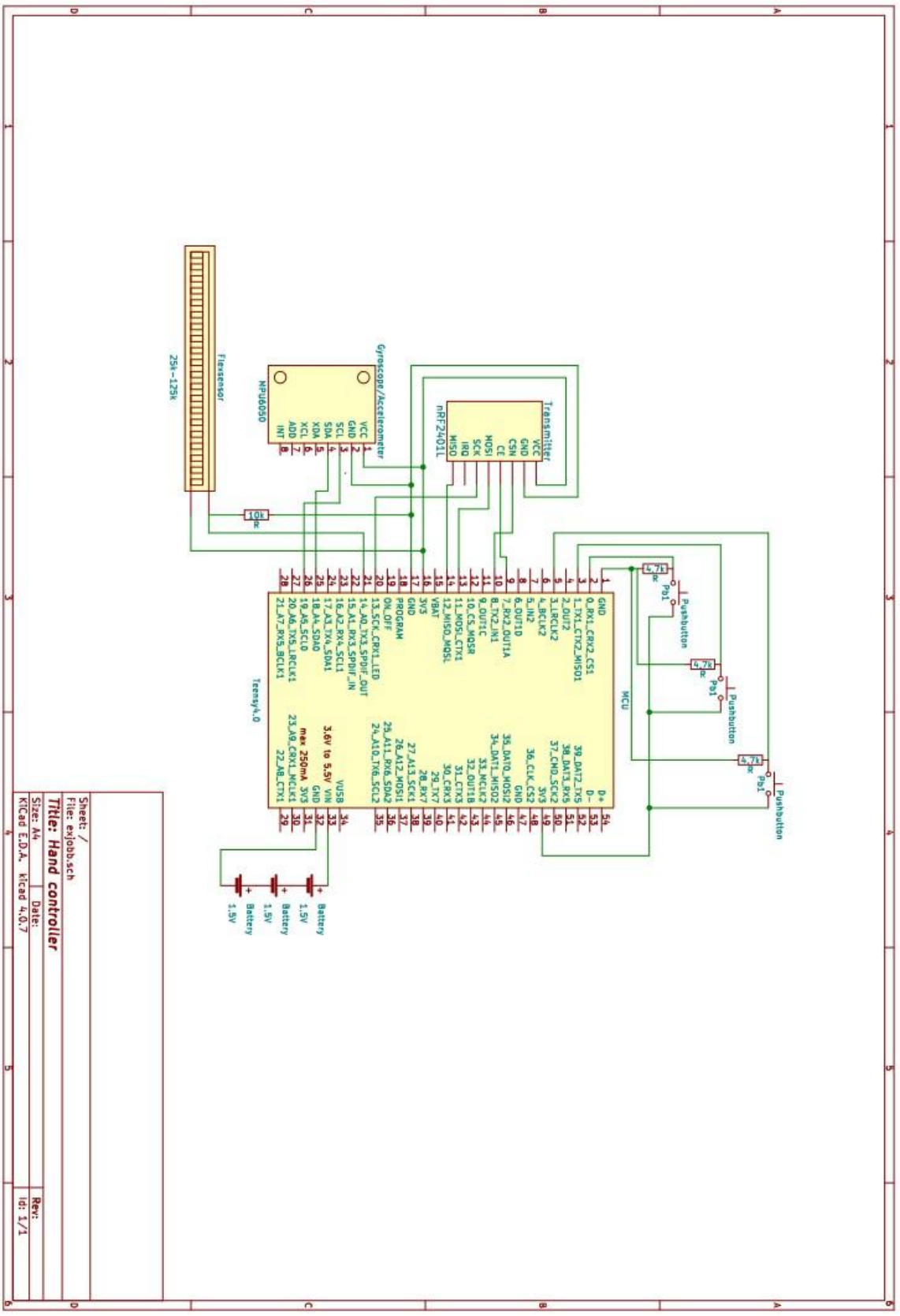
Bilaga G - Kod för mottagare

```
/*

The MIT License (MIT)

Copyright (c) 2022 Bolder Flight Systems Inc

Permission is hereby granted, free of charge, to any person obtaining a copy of this software
and associated documentation files (the “Software”), to deal in the Software without
restriction, including without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
*/

#include <SPI.h>
#include <RF24.h>
#include <sbus.h>

//pins för nRF2401L
#define CE 7
#define CSN 8

bfs::SbusTx sbus(&Serial1); //Tx objekt används för skicka sbus via serial1 pin till drönaren
bfs::SbusData data; //Dataobjekt för paketet som ska skickas via tx

RF24 radio(CE, CSN);
int signals[5];

int* roll_p = &signals[0];
int* pitch_p = &signals[1];
int* yaw_p = &signals[2];
int* throttle_p = &signals[3];
int* arm_p = &signals[4];

const byte address[6] = "ZMART";  //samma adress som sändaren.

bool lostConnection = false;
```

```
uint32_t lastSent;

void setup() {
  setupRadio();
  sbus.Begin();
  delay(1000);
}

void loop() {
  if(!lostConnection){  //om connection tappas en gång, skicka inte nya signaler till drönaren
  if(readRadio()){  //om paket mottogs, skicka vidare paketet till drönaren, spara tiden när
senaste paketet mottogs.
    if(Serial1.availableForWrite()>=(int)sizeof(sbus.data())){
    writeToSbus();
    lastSent = millis();
    delay(15);
    }
  }
  else if(millis()>lastSent+500){ // om inget paket togs emot på 500 ms, stabilisera drönaren
och disarma.
    *roll_p = 993;
    *pitch_p = 993;
    *yaw_p = 993;
    *throttle_p = 700;
    writeToSbus();
    delay(300);
    *arm_p = 500;
    writeToSbus();
    lostConnection = true;  //sluta att läsa radio och ge signaler till drönaren.
  }
  }
  delay(20);
}

//om ett paket hämtas returnera true.
bool readRadio(){
  if(radio.available()){
    radio.read(signals, sizeof(signals)); // paketet sparas i variabeln signals
    //printRadio();
    return true;
  }
  return false;
}

//sätt nrf2401L i receiver mode
void setupRadio(){
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_LOW);
  radio.startListening();
```

```
}

//funktion för att skriva ut signaler.
void printRadio(){
  Serial.println("Radio received!");
  Serial.print("Roll: ");
  Serial.println(*roll_p);
  Serial.print("Pitch: ");
  Serial.println(*pitch_p);
  Serial.print("Yaw: ");
  Serial.println(*yaw_p);
  Serial.print("Throttle: ");
  Serial.println(*throttle_p);
}

//hjälpfunktion för att skriva till sbus
void writeToSbus(){
  data.ch[0] = *roll_p;
  data.ch[1] = *pitch_p;
  data.ch[2] = *throttle_p;
  data.ch[3] = *yaw_p;
  data.ch[4] = *arm_p;
  sbus.data(data);

  sbus.Write();

  delay(10);
}
```

Bilaga H - Kretsschema handkontroller



Bilaga I - Kretsschema mottagare