CHALMERS
UNIVERSITY OF TECHNOLOGY

Autonom informationsinhämtning av rörliga objekt med quadcopter

SSYX02-85 - Kandidatarbete RAM 2016

Slutrapport
16 maj 2016

Sam Halali, Niklas Ohlsson, Anna Samuelsson, Patrik Wallin

Handledare: Patrik Bergagård
Examinator: Martin Fabian

Instutitionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Kandidatarbete 2016

Autonom informationsinhämtning av rörliga objekt med quadcopter

SAM HALALI
NIKLAS OHLSSON
ANNA SAMUELSSON
PATRIK WALLIN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Instutitionen för Signaler och System
Chalmers Tekniska Högskola
Göteborg, Sverige 2016

Autonom informationsinhämtning av rörliga objekt med quadcopter

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

Kandidatrapport 2016:85
Institutionen för Signaler och System
Chalmers Tekniska Högskolan
SE-412 96 Göteborg
Telefon +46 31 772 1000

Framsida: AR.Drone 2.0 Power Edition tillverkad av företaget Parrot. Publicerad med tillstånd av företaget Parrots svenska importör Ingeniörsfirman M Sjöberg AB.

Göteborg, Sverige 2016

Förord

Stora delar av vårt projekt bygger vidare på ett projekt gjort av Markus Lindelöw, Henrik Olsson, Dominik Widmann och Lukas Wikander. Vi är tacksamma för det arbetet de har genomfört och för den hjälp de har bidragit med till vårt projekt.

Vi vill också tacka vår projekthandledare Patrik Bergagård för hans engagemang och vilja i att driva projektet framåt under våra handledningstillfällen. För de givande motfrågor och diskussioner som har uppkommit under dessa tillfällen vilket har lett oss i rätt riktning under hela projektet.

Slutligen vill vi även tacka Calle Carlsson, Magnus Gustafsson och Hans Malmström på avdelningen för fackspråk och kommunikation på Chalmers för deras konstruktiva kritik och värdefulla synpunkter på vår rapport.

Sam Halali
Niklas Ohlsson
Anna Samuelsson
Patrik Walin

Göteborg, 16 maj 2016

Sammandrag

En quadcopter är en variant av UAV (Unmanned Aerial Vehicle) som i dagsläget har många olika användningsområden. En quadcopter styrs oftast manuellt med hjälp av en fjärrkontroll. För att utvidga användningsområdena och frigöra mänskliga resurser utvecklas autonom styrning för quadcoptrar. Autonom styrning sker ofta genom att en dator genererar styrkommandon till quadcoptern baserat på dess sensordata. Detta projekt är influerat av ROAR-projektet där en quadcopter flyger autonomt för att detektera soptunnor. Användningsområdet ska nu vara en byggarbetsplats där quadcoptern används för att övervaka hur olika fordon rör sig. Detta skulle kunna användas för logistik planering eller för att undvika potentiella krockar.

Den här rapporten är resultatet av ett kandidatarbete utfört vid Chalmers Tekniska Högskola. Syftet med detta projekt är att med hjälp av kameror ombord på en autonomt flygande quadcopter bestämma var olika markörer befinner sig samt deras hastighet. Resultatet ska presenteras i ett webbgränssnitt där de olika markörernas position och rörelser visualiseras.

Utveckling av systemet har skett i ROS (Robot Operating System) genom att kombinera existerande bibliotek och arbeten med egenskriven kod. Vid testning har samtliga flygningar skett i inomhusmiljö för att undvika påverkan av väder och vind.

De uppsatta projektmålen uppfylldes delvis. Stora problem inträffade i quadcopterns egna positionsbestämning vilket resulterade i att alla test kopplade till flygning blev lidande. Vidare utvecklades ett webbgränssnitt som ansågs välutvecklat och lättförståeligt av en tillfrågad panel.

Resultaten har visat på att systemet, som inkluderar referensmarkörer för förbättrad positionsuppskattning och detektering av rörliga markörer, fungerar. Dessa skulle kunna användas tillsammans med ett nytt positioneringssystem för att utveckla systemet ytterligare och göra det mer robust och tillförlitligt.

Nyckelord: UAV, Quadcopter, autonom, markördetektering, ROS, AR.Drone, tum_ardrone

Abstract

A quadcopter is a type of UAV (Unmanned Aerial Vehicle) which presently has various applications. A quadcopter is often manually controlled with the assistance of a remote control. To further broaden the possible applications as well as to free human resources, autonomous flight is developed. Based on sensor data from the quadcopter, computer generated control signals are sent to the quadcopter, enabling autonomous flight. This project is influenced by the ROAR-project in which an autonomously piloted quadcopter detects garbage bins. The project explores the application where the quadcopter could monitor the movement of vehicles on a construction site. The purpose of this application could be collision avoidance or logistical planning.

This report is a bachelor's thesis written at Chalmers University of Technology. The purpose of this project is that with the help of cameras on-board an autonomously flying quadcopter, determine different markers' location and velocity. This will be presented in a web based interface where the different markers' location and movement are presented.

The system has been developed in ROS (Robot Operating System) by combining existing libraries and self-produced code. The test environment has been indoors to avoid disturbance from elements such as rain and wind.

The goals of the project was partially accomplished. The quadcopter had complications with estimating its own position which resulted in poor flight and loss of results. Furthermore, a web based interface was developed to display the positions of the quadcopter and the detected markers in a graph. A panel of evaluators found the interface well developed and easy to understand.

The results have shown that the system's use of reference markers, which improves the quadcopter's position estimate, and detection of moving markers is functional. This could be used together with a new positioning system to develop the system further and to make it more reliable.

Keywords: UAV, Quadcopter, autonomous, markerdetection, ROS, AR.Drone, tum_ardrone

Innehåll

Figurer

Tabeller

1
Introduktion

I detta kapitel presenteras den bakgrund projektet bygger på med koppling till samhället idag och forskning inom området. Kapitlet presenterar också begrepp, syfte, projektmål och avgränsningar.

1.1 Begrepp och beteckningar

AR - Augmented Reality
FPS - Frames Per Second
Hovra - Med att hovra menas att quadcoptern håller sin position i x-, y- och z-led under flygning.
Kalmanfilter - Filter för att uppskatta tillstånd hos dynamiska system
N/A - Not Available
Pitch - Rotation kring quadcopterns x-axel
PTAM - Parallell Tracking And Mapping
QVGA - Quarter Video Graphics Array (320x240)
Roll - Rotation kring quadcopterns y-axel
ROS - Robot Operating System
SDK - Software Development Kit
SLAM - Simultaneous Localization And Mapping
Tooltip - Inforuta som syns då man för muspekaren över ett visst objekt
UAV - Unmanned Aerial Vehicle
Yaw - Rotation kring quadcopterns z-axel

1.2 Bakgrund

UAV:er (Unmanned Aerial Vehicle) har funnits sedan början av 1900-talet, men inte förrän på 1970-talet fick användningen ett riktigt uppsving, främst för militärt bruk och senare för besprutning av odlingar [1]. Idag skiljer sig användningsområdena för olika UAV:er drastiskt. De används exempelvis för att underlätta inspektion av svåråtkomliga områden inom fastighetsförvaltning och för att skapa attraktiva bostadsfilmer för mäklare. Med en värmekamera som extra utrustning kan även vattenläckage och andra typer av fel i bostäder enkelt detekteras [2]. Den kan även utnyttjas av polisen för att hitta försvunna personer, men även för att hitta och kartlägga olika djurarter i skogsmiljö för jägare och skogsägare [3]. När mer avancerade

UAV:er tar in och processar mer information utvidgas också användningsmöjligheterna. Utvecklingen av autonom styrning av UAV:er ger bredare användningsområden då man inte behöver en pilot som styr, utan UAV:en sköter arbetet helt på egen hand. Det frigör resurser som kan användas till annat och många gånger kan en dator styra en UAV med högre precision än vad en människa kan.

När det kommersiella bruket av UAV:er ökar uppkommer en rad frågor kring integritet, bland annat att personer känner sig övervakade [4]. Att sprida filmer och foton på människor som inte godkänt det är förbjudet enligt kameraövervakningslagen. Förespråkare anser då att personers integritet kommer att skyddas eftersom det är olagligt att sprida bilder utan personernas godkännande. Det finns också bestämmelser som reglerar flygning med UAV:er, exempelvis att förhålla sig till flygplatsernas kontrollzoner samt att inte flyga över restriktionsområden såsom kärnkraftverk och fängelser [5]. Utöver detta krävs vid kommersiellt bruk tillstånd från transportstyrelsen.

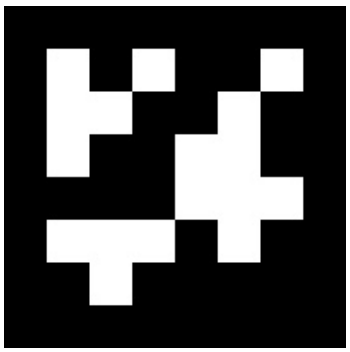
En quadcopter är en variant av en UAV och det som karaktäriserar quadcoptern är dess relativt lilla storlek och fyra propellrar. Quadcopterns historia är relativt ny, men det finns många idéer gällande användandet av dem, inte bara för hobbyflygning [6], [7]. Sedan tidigare finns forskning gällande både att styra quadcoptrar till bestämd flygrutt inom ett visst område och undvika kollisioner [8] såväl som för quadcoptrar att detektera och identifiera objekt [7], [9].

Detta projekt fokuserar på att utöka användningsområdena för autonomt flygande quadcoptrar. Det bygger på tidigare arbeten om autonom styrning av quadcopter och identifiering av objekt med hjälp av markörer. Kod kommer att återanvändas från tidigare arbeten så som *Marker Detection and Autonomous Flight Using Quadcopter Mounted Cameras*, hädanefter kallat *masterprojektet*, där autonom detektering av statiska markörer utförts [10]. Det projektet syftade till att bidra med en del i utvecklingen av ROAR-projektet. ROAR-projektet är ett samarbete mellan Volvo, Renova och ett antal högskolor och universitet som syftar till att automatisera sophämtningsprocessen. Masterprojektets del var att utveckla ett sätt att identifiera och positionera soptunnor i ett område med hjälp av en quadcopter.

Tanken med att detektera rörliga objekt grundar sig i ett intresse från Volvo Group för att kunna komplettera den information man kan hämta från en GPS vid till exempel en byggarbetsplats. Detta skulle till exempel kunna användas för att undvika krockar eller övervaka och effektivisera arbetet. Om markörer finns på alla fordon, eller om en bättre bildhanteringsalgoritm implementeras som kan utskilja dessa fordon utan markörer, kan detta användas både som komplement då en GPS i ett fordon slutar fungera eller för att även kunna ta fordon utan GPS i beaktning.

För att klara av att detektera markörer krävs avancerad bildanalys. Bildanalys betyder att ett datorprogram hämtar information ur en bild, det kan till exempel vara att läsa en QR-kod eller som att känna igen en persons ansikte. Idag används bildanalys inom en rad områden, bland annat kollisionsundvikning för industrirobotar

och inom medicin för att hitta skadliga tumörer vid undersökning. Bildanalys kan också användas för att uppskatta avstånd eller positionering, vilket är hur bildanalys används i detta arbete. Till detta används markörer och ett bibliotek som heter ArUco som är speciellt utvecklat för att känna igen dessa markörer, se Figur 1.1 [11].



Figur 1.1: ArUco-markör. Publicerad med tillstånd från Rafael Muñoz Salinas, en av utvecklarna av ArUco.

Inom markördetektering har arbeten som [12] gjorts. De presenterar en metod för att autonomt hitta och beräkna positioner på markörer utifrån kameran, med hjälp av ett bibliotek kallat ArUco. Vid det tekniska universitetet i München har avhandlingar om hur en lågkostnads-quadcopter navigerats endast med hjälp av medföljande sensorer och kameror publicerats [13].

För att enkelt kunna visualisera var markörerna befinner sig finns en rad olika metoder. Genom att förmedla information på en webbsida blir den lätt tillgängligt för många. Idag går det att komma åt webbsidor från många olika enheter och platser, så länge det finns internetuppkoppling.

Då mycket forskning om nya användningsområden för quadcoptrar finns eller är på gång är inte målet med projektet i första hand att kunna presentera revolutionerande resultat gällande autonom markördetektering. Det är snarare för att ge kandidatgruppen en insikt i hur ett projektarbete som detta kan utföras och ett tydliggörande i hur autonom informationshämtning kan gå till samt för att utforska ett potentiellt användningsområde.

1.3 Syfte

Syftet med detta projekt är att med hjälp av kameror ombord på en autonomt flygande quadcopter bestämma var olika markörer befinner sig samt deras hastighet. Resultatet ska presenteras i ett webbgränssnitt där de olika markörernas position och rörelser visualiseras.

1.4 Projektmål

Det övergripande målet med projektet är att utveckla masterprojektets användningsområde, vilket var ROAR-projektet. De har med autonom informationsinhämtning positionerat statiska markörer med viss felmarginal [10]. Detta utvecklas nu till att identifiera rörliga markörer och utföra hastighets- och positionsberäkningar med informationen som samlas in. Flygdata, såsom resultat av beräkningarna och quadcopterns position, uppdateras och presenteras i realtid i ett webbgränssnitt. Målet har delats upp i ett antal delmål.

1.4.1 Återskapa masterprojektets resultat med existerande bibliotek

Då ett omfattande arbete gällande autonom detektering av statiska markörer redan har gjorts, utgör det första delmålet att återskapa tidigare resultat med hjälp av existerande bibliotek för att kunna bygga vidare på detta.
Med tidigare resultat menas:

- Beräkning av flygrutt baserat på ett fördefinierat sökområde.
- Autonom flygning av beräknad flygrutt
- Detektering av statiska markörer
- Positionering av markörer med en felmarginal på maximalt 0,5 m [10].

1.4.2 Implementering av referensmarkörer

Masterprojektet konstaterade att markörpositioneringsalgoritmerna fungerar bra. Det är på grund av att quadcoptern "tappar bort" sig själv i rummet som större delen av felet i markörpositioneringen uppstår. Quadcopterns felmarginal i sin egen positionering måste därför förbättras.

Då fel troligtvis kommer att kvarstå även om punkt 1.4.1 utförs med goda resultat införs referenspunkter i rummet. Med referenspunkter menas att markörer med en känd position placeras ut inom det angivna området, vilket gör att quadcoptern kan kalibrera sig efter dessa under flygning. Med anledning av quadcopterns problem med sin egen positionering sätts målet till att förbättra felmarginalen med 10% jämfört med resultatet av mål 1.4.1.

1.4.3 Detektering av rörliga markörer

Dessutom ska quadcoptern nu detektera och positionera rörliga markörer med samma felmarginal som för statiska markörer i masterprojektet.

1. Första delmålet är att detektera rörliga markörer med samma felmarginal som för en stilla markör, då den filmas av en kamera som är stilla. Quadcoptern är

alltså statisk under detta test.

2. När systemet detekterar markörer när quadcoptern står stilla utvidgas målet till att både markörerna och quadcoptern rör sig. Som innan ska samma felmarginal som när markörerna var stilla uppnås.

1.4.4 Webbgränssnitt för presentation av markörer

För att användaren enkelt ska få en överblick över markörernas tillstånd utvecklas ett webbgränssnitt. Detta gränssnitt visar en graf över markörens senast kända position och även dess senast beräknade hastighet. Webbgränssnittet läser i realtid från en databas som fylls på med inhämtad information. Målet är att gränssnittet tydligt presenterar markörernas senast kända position samt hastighet. En panel med testpersoner tillfrågas om tydligheten på webbgränssnittet. Om medelbetyget panelen ger är minst *3*, på en skala *1* till *5*, anses målet uppnått.

1.5 Avgränsningar

För detta projekt kommer quadcoptern att flygas inomhus för att eliminera felkällor så som väder och vind. Avsökningsområdet, plant golv, som quadcoptern söker av kommer att vara statiskt och bestämmas på förhand. Då alla testkörningar sker inomhus och inom ett förbestämt område minskar risken för oväntade krockar. Med förankring i bakgrunden, att denna typ av informationsinhämtning skulle kunna användas på en byggarbetsplats, är det inte heller aktuellt att ha någon annan typ av sökområde då dessa är statiska och fördefinierade. Då quadcoptern inte är utrustad med GPS kommer fastställandet av markörernas position endast ske med hjälp av de inbyggda kamerorna och sensorerna. Detta leder till att positionen endast är känd relativt det förbestämda sökområdet.

Även andra typer av avgränsningar tillkommer på grund av quadcopterns hårdvarubegränsningar. Till exempel är höjden quadcoptern max kan flyga på, och fortfarande använda sig av sin ultraljudssensor, sex meter [4]. Quadcoptern har två kameror men kan bara skicka bilder från en av dem i taget [10]. Under tiden datorn får bilder från bottenkameran, som ska hitta markörer, kommer framkameran, som ska hålla positionen i rummet, inte att få några bilder. Det kommer att begränsa frekvensen bilderna kan skickas med då masterprojektet visat att man måste skicka ett visst antal bilder för att försäkra sig om att någon av bilderna från varje kamera kommer fram. Frekvensen kommer att påverka hastigheten som quadcoptern kan att färdas i, för att inte missa något område under quadcoptern. Hastigheten quadcoptern kan färdas i begränsas även av höjden den flyger på.

Webbgränssnittet kommer att begränsas till att endast kunna visualisera data inhämtad av quadcoptern. Det kommer alltså inte vara möjligt att varken styra quadcoptern manuellt eller att ändra sökparametrar, som sökarea eller avsökningsmetod, från webbgränssnittet.

2

Teknisk beskrivning

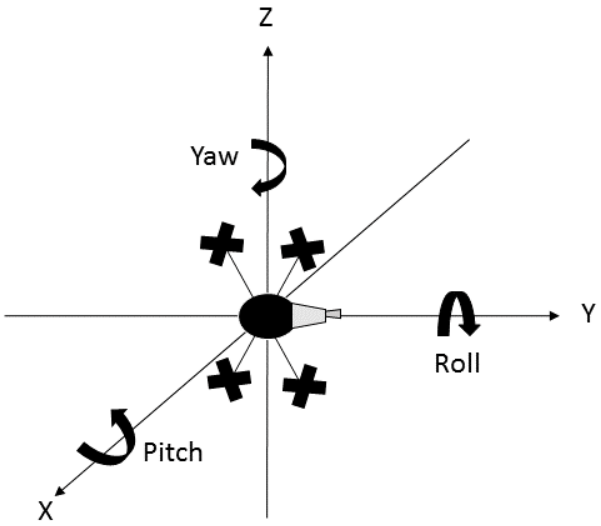
I detta kapitel ges en introduktion till quadcoptrar i allmänhet och quadcoptern som används i projektet. Utöver detta beskrivs hur markördetektering, quadcopterpositionering samt webbgränssnitt fungerar och hur dessa kopplas ihop.

2.1 Quadcopter

En quadcopter är en UAV med fyra motorer monterade på varsin arm som sitter symmetriskt placerade runt quadcopterns kropp. Till varje motor sitter en propeller monterad och de två propellrarna som sitter monterade på samma axel roterar åt samma håll. Hastigheten vilket varje motor körs i kan bestämmas helt individuellt och det är så quadcoptern styrs. Detta genom att varje motor styrs med hjälp av styrsignaler beräknade av ett styrkort som sitter i quadcoptern. För att quadcoptern ska kunna flyga stabilt och kunna *hovra*, hålla samma position i luften, finns även olika sensorer i quadcoptern så eventuella positions- eller vinkeländringar kan registreras och kompenseras för.

Många quadcoptrar styrs idag manuellt med hjälp av någon sorts fjärrkontroll, exempelvis en applikation i mobilen. I quadcoptern finns ett styrkort som med hjälp av olika sensorer kan reglera hur de olika motorerna ska köras för att flyga, kompensera för vind och hålla quadcoptern stabil i luften. Då styrkortet förväntar sig styrsignaler, oberoende av ursprung, kan de genereras av till exempel en dator. Styrsignalerna kan produceras genom att datorn tar emot tillståndsdata från quadcoptern och därefter beräknar de signaler som krävs för att uppnå ett fördefinierat resultat, som att flyga en viss sträcka. Detta möjliggör autonom styrning av quadcoptern.

Quadcopterns tillstånd i rummet bestäms av dess position och rotation. För att beskriva quadcopterns position i rummet används x-, y- och z-koordinater och riktningarna, relativt quadcoptern, beskrivs i Figur 2.1. För att beskriva rotationen används vinklarna *pitch*, *roll* och *yaw* som definieras av rotationen kring quadcopterns x-, y- och z-axel, se Figur 2.1.



Figur 2.1: Figur över quadcopterns koordinatsystem.

2.2 AR.Drone 2.0

Quadcoptern som används i detta projekt är av modell AR.Drone 2.0 från företaget Parrot. På quadcoptern finns en mängd olika sensorer som underlättar reglering och navigering. Det finns två kameror, en fram som filmar i HD (720 p) och 30 *FPS* (Frames Per Second - rutor per sekund) och en undertill som filmar i *QVGA* (Quarter Video Graphics Array, 320x240) och 60 FPS. Framkamran har en vidvinkellins på 92° diagonalt. Quadcoptern kan bara skicka bilder från en kamera åt gången. Nedan listas de sensorer som quadcoptern i fråga är utrustad med, samtliga hämtade från produktsidan för AR.Drone 2.0 [14]:

- Ett 3-axligt gyroskop som används för att mäta quadcopterns lutning i x-, y- och z-led. Denna data kan sedan användas för att reglera motorerna så quadcoptern hovrar stabilt. Gyroskopet har en precision på 2000° per sekund.
- En accelerometer med 3 axlar som mäter quadcopterns egenacceleration med en precision på ± 50 mg.
- Magnetometer som bestämmer quadcopterns position i förhållande till jordens magnetfält med en precision på 6°.
- Ultraljud som används för att estimera höjden quadcoptern befinner sig på. Genom att skicka ut ett ultraljud och låta det studsa i marken, tillbaka till quadcoptern, kan avståndet beräknas genom att mäta tiden detta tog.
- Ultraljudssensorn fungerar inte vid höjder större än sex meter [4]. Därför är quadcoptern även utrustad med en barometer för att estimera höjden. Barometern har en noggrannhet på +/- 10 Pa.

AR.Drone 2.0 har möjligheten att styras med hjälp av en applikation i mobilen och kan även styras av en dator genom att ta emot styrdata genererad av den. Styrdatan är vektorer med x-, y-, z-koordinater samt rotation i yaw-led och skickas över Wifi.

2.3 ROS

ROS (Robot Operating System) är ett ramverk av bibliotek och verktyg utvecklade för robotutveckling. ROS har tagits fram då det är svårt att utveckla robust programvara för robotar [17]. Vanliga uppgifter, som skulle vara triviala för oss människor, blir fort mycket komplexa för robotar [17].

Ett ROS program består av många olika delprocesser, kallade noder, och kommunikationen mellan dessa kan ske med två olika metoder. Antingen via en Publish/Subscribe-modell eller via en Request/Response-modell.

I Publish/Subscribe-modellen publicerar eller hämtar varje nod data. Noderna som publicerar kallas för publisher och noderna som hämtar data kallas subscriber. En nod kan vara både publisher och subscriber. Där dessa noder publicerar eller hämtar data från kallas för en topic och kan ungefär ses som en anslagstavla. En publisher skickar data till en topic, sätter upp något på anslagstavlan, och en subscriber tar emot data från en topic, läser vad som finns på anslagstavlan. Värt att notera är att subscribers hämtar datan direkt när den publiseras på topicen. Både publishers och subscribers kan även arbeta med flera topics samtidigt, men, det finns ingen garanti att någon nod hämtar data från en topic bara för att en annan publicerar till den. Strukturen kan även beskrivas som en graf, där varje nod i grafen antingen är en publisher eller en subscriber och kanterna mellan noder är topics.

I Request/Response-modellen sker kommunikationen mellan noder genom tjänster, på engelska services. Enligt [18] beskrivs modellen följande. Noderna delas upp i klient-noder och server-noder. En klient-nod skickar data till en server-nod och väntar sedan på ett svar, detta kallas för en förfrågan, på engelska request. Server-noden tar emot förfrågan och utför tjänsten som efterfrågas. Tjänsten kan innefatta uträkningar, konfiguration av hårdvara/mjukvara eller liknande. Datan som sedan skickas tillbaka till klient-noden kallas för ett svar, på engelska response. Med Request/Response-modellen sker kommunikationen åt båda hållen då varje klient-nod förväntar sig ett svar från server-noden den skickat data till. Detta till skillnad från den andra modellen, där ingen nod behövde hämta data från en topic bara för att det publicerades till den. En annan skillnad är att kommunikationen endast sker mellan en klient-nod och en server-nod, till skillnad från Publish-Subscribe-metoden där flera noder antingen kan skriva eller läsa från fler topics.

Nodstrukturen i ROS gör programmen modulära, vilket förenklar utvecklingen med hjälp av existerande bibliotek och moduler.

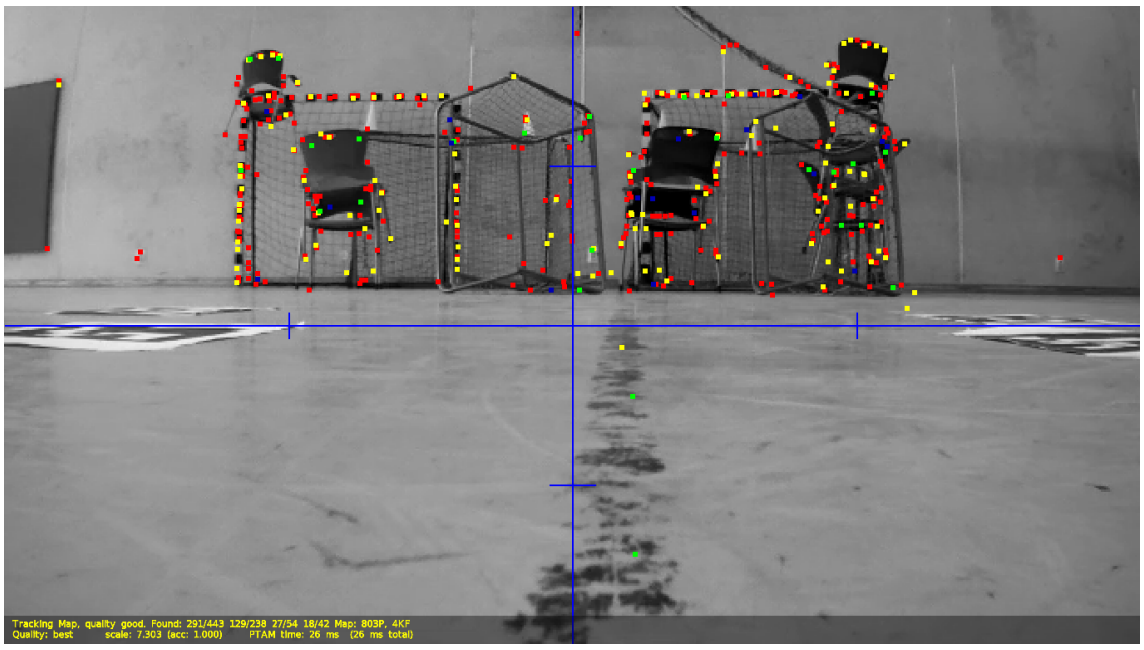
I ROS kan även komponenter skrivna i olika programspråk kombineras. De stödda språken är: Python, C++ och Lisp [17]. Då gruppen främst har kompetens i Python och för att minimera *Lines of Code* sker utvecklingen i Python. Med Lines of Code menas antal rader kod som krävs för att uppnå samma resultat hos de olika programspråken. Då AR.Drone-plattformen är kompatibel med ROS och då ROS har många befintliga bibliotek och möjliggör för modulär mjukvaruutveckling kommer

ramverket att användas i projektet.

2.4 Positionsuppskattning för quadcopter

För att quadcoptern ska kunna bestämma sin position i rummet kommer ett *SLAM-system* (Simultaneous Localization And Mapping) att användas. SLAM-system använder sig av en robot för att utforska och kontinuerligt generera en karta över ett tidigare okänt område [15]. Denna karta används senare för att navigera i området [15].

Det SLAM-system som används kallas PTAM (Parallell Tracking And Mapping). PTAM använder sig av bilder skickade från en kamera för att generera nyckelpunkter från objekt i rummet som sedan används för navigering. Dessa nyckelpunkter skapas då två bilder, på olika höjd, tas med framkameran på quadcoptern. De tvådimensionella bilderna skapar tillsammans ett tredimensionellt rum vilket ses i Figur 2.2. Detta betyder att quadcoptern inte behöver några markörer, fördefinierade kartor eller inbyggda sensorer för att kunna uppskatta sin position [16].



Figur 2.2: Visualisering av PTAMs nyckelpunkter. Figuren är tagen av författarna med hjälp av mjukvaran tum_ardrone.

En nackdel med att använda PTAM är att nyckelpunkter endast samlas in under initialiseringsprocessen. Detta kan leda till att olika insamlade punkter med liten djupskillnad uppfattas fel och därmed sänks PTAM-prestandan. Då precisionen i quadcopterns uppskattade tillstånd till stor del beror på bra PTAM-prestanda är det därför viktigt att göra en bra PTAM-initiering med många nyckelpunkter för att få en noggrann positionsuppskattning [10].

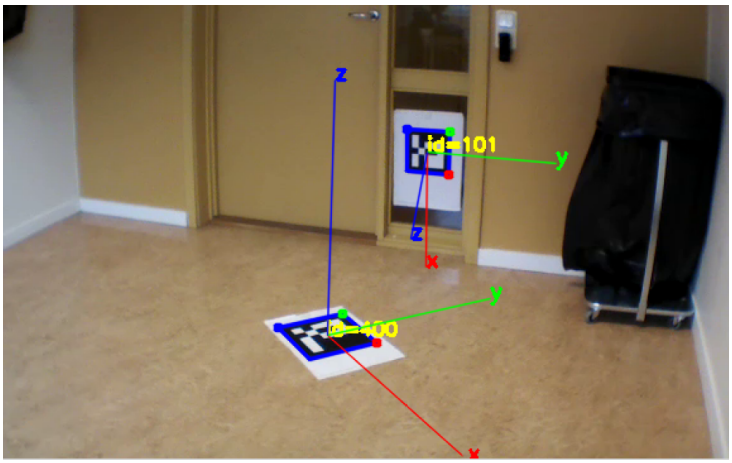
2.5 Markördetektering

För identifiering av markörer används ett redan existerande bibliotek. Detta bibliotek heter ArUco och är speciellt utvecklat för att känna igen svart-vita fyrkantiga markörer.

Hur dessa markörer är uppbyggda, vad för information som lagras i dem samt hur detekterings- och positioneringsprocessen av dem går till beskrivs enligt [26] som följande. Markörerna består av en binär matris omgiven av en tjock, svart kant. I en binär matris finns endast booleska värden (0, 1) och dessa representeras av svarta och vita områden innanför kanten. Den binära matrisen möjliggör även för feldetektering och korrektion av fel. Det är på detta vis identifikationen (ID) för varje markör lagras i matrisen.

Själva markördetekteringen i en bild är sedan uppdelad i två steg. Först analyseras bilden genom att leta efter fyrkantiga figurer som skulle kunna vara markörer. Detta görs med en rad olika bildbehandlingstekniker. När denna process är klar filtreras vissa av dessa figurer bort genom att analysera varje figurs matris. Om matrisen är oläslig filtreras denna figur bort och kvar finns endast de detekterade markörerna tillsammans med respektive ID.

Utifrån dessa identifierade markörer kan sedan varje markörs position, relativt kamerans koordinatsystem, bestämmas. Varje markör har sitt egna koordinatsystem där mittpunkten på markören är i origo. Med hjälp av en rotations- och en translationsvektor kan markörens koordinatsystem transformeras till kamerans koordinatsystem och på så vis kan en position på markören utläsas. I Figur 2.3 visas hur två markörer med olika koordinatsystem och ID detekterats.



Figur 2.3: Två detekterade markörer med respektive markörs koordinatsystem. Figuren skapad av författarna med hjälp av ArUco.

2.6 Utveckling av webbgränssnitt

För att lätt kunna komma åt informationen om hur markörerna rör sig från olika enheter, utan att behöva installera program, utvecklas ett webbgränssnitt. HTML är ett språk utvecklat för att visa text och grafik över Internet, som ett dokument som visas av webbläsaren [19]. Med hjälp av CSS, som är ett språk som används för att beskriva hur dokumentet skrivet i HTML ska se ut, kan en enkel webbsida byggas upp [20].

Webbsidan kan göras mer komplex med hjälp av JavaScript [21]. JavaScript är ett språk som ofta används tillsammans med HTML och CSS för att utveckla webbsidor, enligt hemsidan javascript.com [22]. Enligt samma källa fungerar det så att användaren som vill se hemsidan laddar ner dels, HTML-koden, men också JavaScript-koden som exekveras på användarens dator. JavaScript används för att göra komplexa uträkningar eller funktioner i en webbläsare och är idag ett viktigt verktyg för att skriva dynamiska webbsidor [21].

För att samla och lagra all information webbgränssnittet ska visa används en databas. En databas är en organiserad samling data, idag ofta digital. En digital databas kan ses som en samling tabeller som man med hjälp av speciella kommandon kan hämta relevant data från. SQL är ett språk som kan användas för att lagra, hämta och manipulera data i databasen, men för att skriva applikationer som använder sig av datan behövs andra språk som C++ eller PHP [23]. PHP är ett språk som kan användas för webbutveckling och kan kombineras med HTML [24].

Från databasen läses sedan information som ett JSON-objekt. Ett JSON-objekt är ett format för att byta data mellan olika programmeringsspråk. JSON-objekt skrivs med JavaScript-syntax men är språk-oberoende så att många olika programmeringsspråk kan använda sig av dem [25].

3
Mjukvara

För att få en djupare förståelse om mjukvaran i projektet presenteras här hur de olika delarna är uppbyggda samt hur de kommunicerar med varandra. I kapitlet redogörs även för olika paket och bibliotek som används och hur de kombineras.

3.1 Bibliotek som används

Programmet som används för autonom styrning och markördetektering består till stor del av redan skriven kod. Som tidigare nämnts tillåter ROS att många olika bibliotek används och vävs samman. Nedan presenteras biblioteken som används i projektet.

OpenCV är ett bibliotek med bildbehandlingsalgoritmer som används för detektering av markörer såväl som för positionering av quadcoptern. Biblioteket utvecklades för att användas i realtids-applikationer med fokus på att vara beräkningseffektivt [26]. Med hjälp av vision_opencv, som är ett paket för ROS, kan OpenCV integreras och användas i ramverket. Detta med hjälp av en brygga som översätter meddelanden mellan OpenCV och ROS [27]. Ett bibliotek som bygger på OpenCV är ArUco. Det är utvecklat för att i realtid kunna känna igen en speciell typ av markörer, kallade ArUco-markörer, och bestämma deras positioner [28]. ArUco-biblioteket används för att bestämma var markörerna befinner sig i förhållande till kameran, alltså i förhållande till quadcoptern i detta fall. Även ArUco kan integreras med ROS.

Ett annat ROS-paket som används är ardrone_autonomy som är utvecklat för AR.Drone 1.0 och AR.Drone 2.0 baserat på Parrots officiella *SDK* (Software Development Kit) för AR.Drone [28]. Detta paket används för att skicka navigeringsdata till quadcoptern och för att ta emot film från kamerorna och sensordata från de olika sensorerna [29]. Frekvensen data skickas med kan sättas till antingen 15 eller 200 Hz. För att styra quadcoptern publiceras meddelanden med x-, y-, z- och yaw-värden till en topic som sedan läses av ardrone_autonomy som i sin tur skickar styrsignaler till quadcoptern [30].

För att möjliggöra navigation med quadcopterns framkamera används ROS-paketet tum_ardone. Detta paket bygger på det tidigare nämnda ardrone_autonomy. tum_ardrone består av tre komponenter; ett SLAM-system som använder sig av en kamera, ett utökat *Kalman-filter* för bland annat tillståndsuppskattning samt en PID-kontroller för att generera styrkommandon [31].

Som tidigare nämnts används och modifieras kod från masterprojektet, programmet roar_quad. Programmet kombinerar tum_ardrone och ArUco med egenskriven kod. tum_ardrone används för att autonomt flyga över ett förbestämt område och ArUco används för att detektera markörer från bilder som tas under flygningen. Utifrån ett angivet sökområde genereras en flygrutt som består av ett antal punkter som skickas punktvis till tum_ardrone. Baserat på sin positionsestimering skickar tum_ardrone styrkommandon till ardrone_autonomy som i sin tur styr quadcoptern. Bilder från quadcopterns bottenkamera tas kontinuerligt emot av roar_quad och skickas till ArUco för detektion av markörer. Med hjälp av markörens position i bilden och tum_ardrones uppskattade position av quadcoptern beräknas en uppskattad position av markören i rummet. Med hjälp av roar_quad kan alltså quadcoptern flyga autonomt över ett förbestämt område och detektera markörer samt beräkna markörpositioner.

3.2 Egenskriven kod

För att uppnå syftet med projektet behövs masterprojektets kod modifieras. Utöver detta krävs att egen kod skrivs för att lägga till önskad funktionalitet. Delar som behövs läggas till är implementation av referensmarkörer samt presentation av data i webbgränssnittet. Nedan redogörs de olika delarnas funktion.

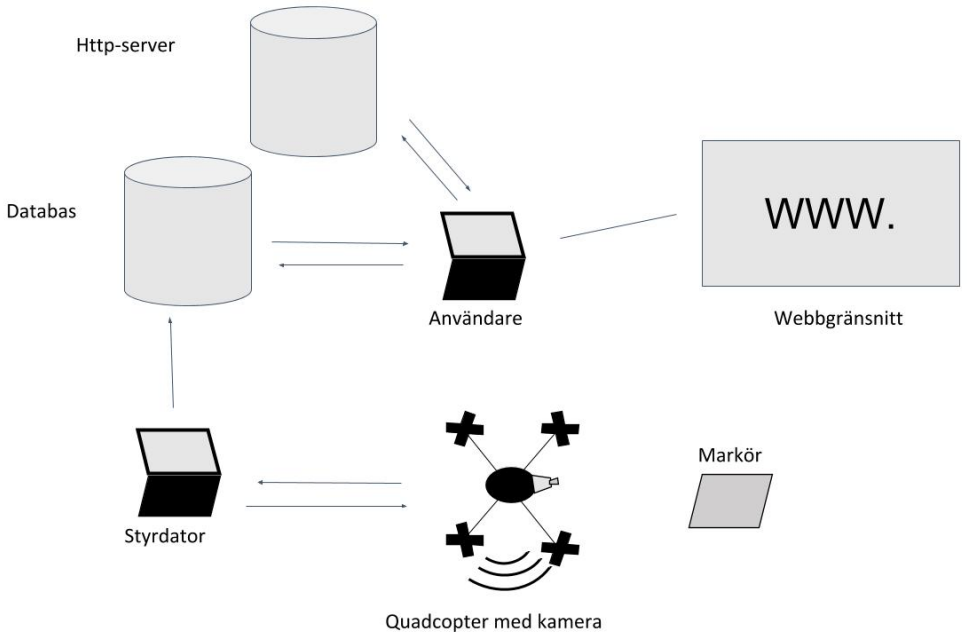
3.2.1 Webbgränssnitt

Webbgränssnittet delas upp i ett presentationslager som består av användargränssnittet och ett datalager. Datalagret består i sin tur av ROS-programmet och en databas. Datalagrets uppgift är att beräkna markör- och quadcopterdata samt att göra denna tillgänglig för presentationslagret. Presentationslagrets uppgift är att hämta markör- och quadcopterdata från datalagret och presentera denna på ett sätt som gör datan överskådlig.

För att göra data från ROS-programmet tillgängligt för utomstående program skrivs egna noder som sköter länken mellan den insamlade datan från körning och databasen. Direkt vid initiering laddar en nod upp en ny session till databasen. Sessionen definieras av ett unikt ID, starttiden för körningen, samt information om sökområdet. Webbgränssnittet ger användaren en lista av unika ID:n som är tillgängliga i databasen. När ett ID väljs presenteras quadcopterns position och de detekterade markörernas positioner för den specifika körningen i en graf. Vid detektion av en markör sköter en annan nod uppladdning av information gällande markör-ID samt markörposition för den detekterade markören till databasen och en tredje nod laddar varje sekund upp quadcopterns aktuella position. Webbgränssnittet frågar kontinuerligt databasen om ny data kopplat till det unika ID:t. Om ny data finns tillgänglig hämtas den som ett JSON-objekt och används för att uppdatera användargränssnittet.

Webbgränssnittet består av en graf där quadcopterpositionen och de detekterade markörerna visas. Information om beräknad markörposition samt markörhastighet finns tillgänglig i en *tooltip*, en inforuta som visas då användaren för musen över en markör. Gränssnittet består även av en inforuta angående den aktuella körningen, en inforuta med information gällande quadcopterns position och hastighet samt en logg där varje detektion av markörer visas med en tillhörande tidsstämpel.

Sammanfattningsvis kan webbgränssnittet förklaras med hjälp av Figur 3.1. Data skrivs kontinuerligt till en databas som kan nås globalt. Användaren av webbgränssnittet begär koden för gränssnittet från en HTTP-server och exekverar sedan det på sin dator. Programmet kommer sedan att kontinuerligt att hämta ny data från databasen och presentera det i webbgränssnittet.



Figur 3.1: Modell över hur data samlas in av quadcoptern, laddas upp i databasen samt presenteras av webbgränssnittet för användaren.

3.2.2 Implementering av referensmarkörer

För implementering av referensmarkörer var idén initialt att quadcopterns position skulle omkalibreras då systemet detekterade en referensmarkör. På grund av att tum_ardrone-paketet inte stödjer detta som förväntat valdes ett annat alternativ. Referensmarkörernas faktiska positioner förprogrammeras i programmet. Då quadcoptern under flygning detekterar en referensmarkör räknar programmet ut var markören befinner sig, likt en vanlig markör. Därefter beräknas felen, x_{error} och y_{error}, mellan var programmet uppskattar referensmarkörens position och dess faktiska position i x- och y-led enligt ekvation 3.1.

$$\begin{aligned} x_{error} &= x_{ref} - x_{uppmätt} \\ y_{error} &= y_{ref} - y_{uppmätt} \end{aligned} \tag{3.1}$$

Det beräknade felen subtraheras sedan från quadcopeterpositionen och från den beräknade markörpositionen varje gång en ny markör detekteras. Detta sker innan positionerna laddas upp i databasen. När en ny referensmarkör hittas uppdateras felet och innan en referensmarkör hittats är felet noll.

3.2.3 Detektering av rörliga markörer

Masterprojektets kod fungerar för att detektera och beräkna positioner för rörliga markörer. Därför görs inga förändringar i koden för implementering av rörliga markörer. Implementeringen sker istället i koden för webbgränssnittet. Där används funktioner för att hämta de två senaste positionerna för både quadcoptern och markörer och räknar därefter ut hastigheten mellan dessa två punkter. Markörernas position visas i tillhörande tooltip i webbgränssnittet och quadcopterns hastighet visas i en ruta bredvid grafen. Om en markör bara detekteras en gång kan ingen hastighetsberäkning utföras och *N/A* (Not Available) visas i tooltipet. Varje gång en redan detekterad markör hittas på en ny position uppdateras dess hastighet samt så informeras användaren om att en markör rört sig genom att uppdatera loggen.

4

Utformning av tester

Starkt kopplat till projektmålen genomförs olika tester för att se om dessa mål har uppnåtts. För de olika projektmålen presenteras motsvarande tester nedan.

4.1 Återskapa masterprojektets resultat med existerande bibliotek

Till en början återskapas resultaten från masterprojektet och testflygningen sker på samma sätt som i det arbetet. Givet ett visst område beräknas en rutt vilken quadcoptern flyger för att detektera markörer. Felmarginal för positionering av markörer får max vara 0,5 m då detta var felmarginalen i positionering från masterprojektet.

4.2 Implementering av referensmarkörer

Två markörer placeras ut inom ett givet område. Därefter initieras PTAM-nyckelpunkter och sedan startas söksekvensen. För att enkelt kunna testa referensmarkörernas påverkan på markörpositioneringen bärs quadcoptern över sökområdet. Då felet i markörpositionering antecknats upprepas testet även med två referensmarkörer utplacerade i området. Felet i markörernas uppskattade positioner med och utan referensmarkörer jämförs.

Test 4.1, som beskrivs ovan, upprepas sedan med referensmarkörer utplacerade och skillnaden i felmarginal med och utan referensmarkörer jämförs.

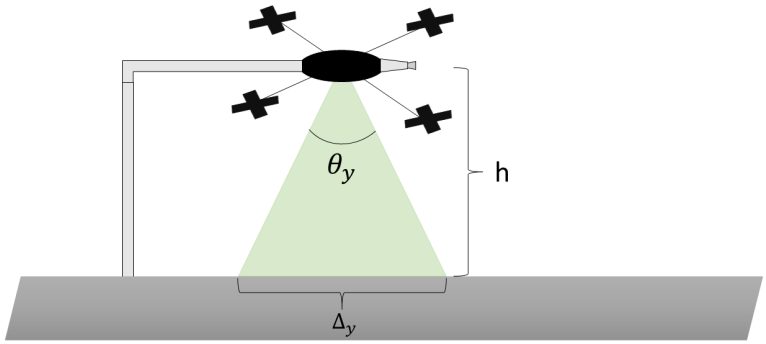
4.3 Beräkning av maxhastighet för rörliga markörer

För att bestämma den högsta hastighet en markör och quadcoptern kan röra sig i samtidigt som minst en detektion av markören görs, behövs arean vilket bottenkameran registrerar mätas. Genom att mäta vinkelupptagningen på bottenkameran räknas den arean ut. För att mäta dessa vinklar sätts quadcoptern fast i en ställning på en känd höjd h, se Figur 4.1a. Därefter mäts avstånden Δx och Δy, vilka utgör kamerans synfält, se Figur 4.1b. Vinklarna Θ_x och Θ_y räknas sedan ut genom ekvation 4.1.

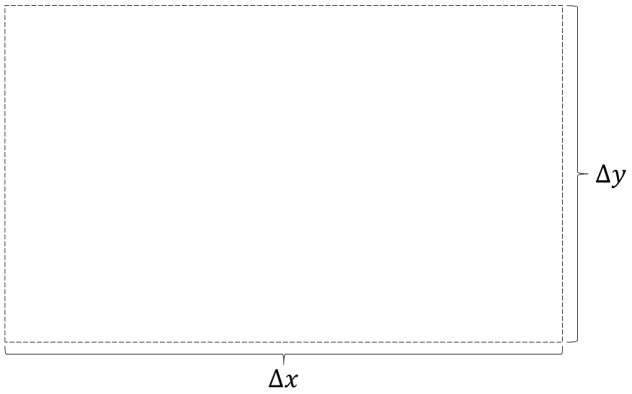
$$\begin{aligned} \theta_x &= 2 * arctan(\frac{\frac{\Delta x}{2}}{h}) \\ \theta_y &= 2 * arctan(\frac{\frac{\Delta y}{2}}{h}) \end{aligned} \tag{4.1}$$

Med dessa vinklar räknas sedan Δy ut vid flyghöjd. För att detektera en markör krävs att hela markören syns och längden för en markör, $l_{markör}$, är 0,3 m. Den längsta tiden vilket programmet inte tar emot bilder från bottenkameran, t_{botten}, är 0,133 s [10]. Med detta räknas sedan maxhastigheten ut genom ekvation 4.2.

$$v_{max} = \frac{\Delta y - l_{markör}}{t_{botten}} \tag{4.2}$$



(a) Illustration av test för vinkelupptagning av bottenkameran



(b) Bottenkamerans synfält

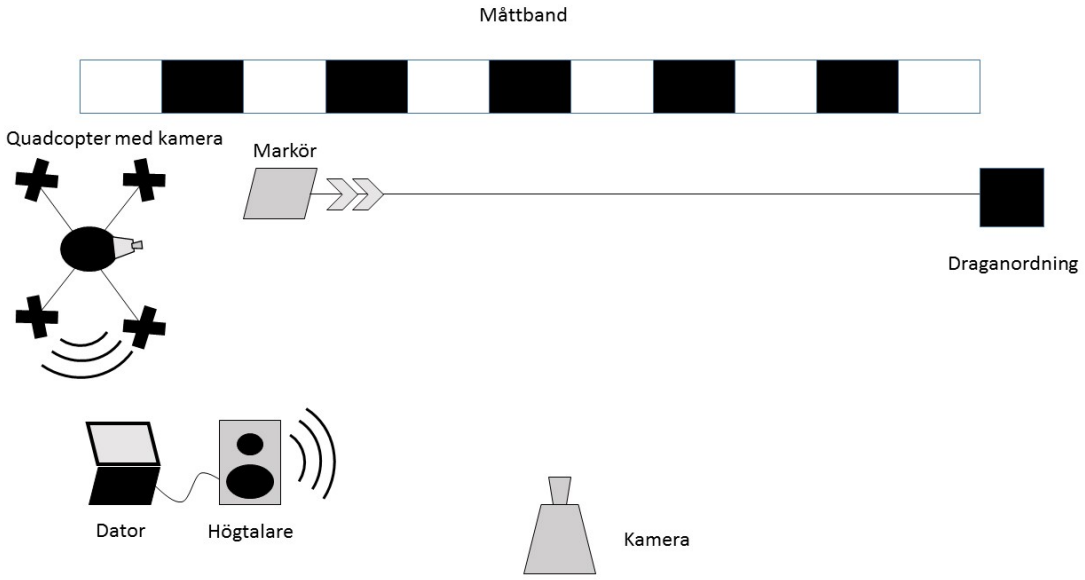
Figur 4.1: Sidovy över quadcopterns placering vid test av bottenkamerans vinkelupptagning samt bottenkamerans synfält.

4.4 Detektering av rörliga markörer

För testning då markörerna rör sig dras de i en rätlinjig rörelse, vinkelrät mot en kamera som filmar. Då quadcoptern detekterar en markör beräknas dess position, samtidigt skickas en signal till en dator vilket gör att en ljudsignal lyder. Markörens faktiska position bestäms sedan genom att kolla vart den befinner sig på måttbandet vid ljudsignalen. Genom att jämföra denna position med quadcopterns uppmätta position av markören bestäms felet i uppmätt positionering av markören.

Ett annat sätt att testa detta är genom att låta den fristående kameran följa ytterligare en markör riktad mot sig. Kameran är direkt kopplad till datorn där både quadcopterns och den externa kamerans bildströmmar jämförs för att detektera skillnaden i uppmätt position.

I testet befinner sig quadcoptern statiskt i rummet för att underlätta markördetektering samt för att minimera fel i quadcopterns egna positionsuppskattning.



Figur 4.2: Illustration av test för icke-statiska markörer.

4.5 Webbgränssnitt

För testning av huruvida webbgränssnittet fyller sin funktion använder en testpanel webbgränssnittet och betygsätter dess tydlighet. Frågor om användarna enkelt kan läsa ut markörens senaste position och hastighet ställs. Användarna betygsätter tydligheten på en skala från *1* till *5*, där målet är att minst få medelbetyget *3*.

5

Testresultat

Nedan presenteras resultaten av de olika tester som genomförts för att verifiera om projektets delmål uppnåtts. Diskussion gällande resultat återfinns i kapitel 6.

5.1 Återskapa masterprojektets resultat med existerande bibliotek

Quadcopter kunde inte flyga samma område som masterprojektet gjorde under sitt test utan att förlora PTAM-nyckelpunkterna. Masterprojektets testområde var 4x6 meter och istället gjordes testet på ett område av 2x3 meter. Markörernas positioner i koordinatsystemet var (0,50; 0,00), (-0,50; 1,00), (0,50; 1,00). Felet i markördetektering ses i Tabell 5.1. Medelfelet för markörerna blev följande: $\bar{e}_1 = 0{,}32$ m, $\bar{e}_2 = 0{,}40$ m och $\bar{e}_3 = 0{,}47$ m.

Tabell 5.1: Mätning av felmarginal på markörpositionering

	Markör 1	Markör 2	Markör 3	e_1	e_2	e_3
Test 1	(0,68; -0,58)	(-0,40; 0,54)	(0,66; 0,33)	0,61 m	0,47 m	0,68 m
Test 2	(0,51; 0,07)	(-0,36; 0,77)	(0,69; 0,72)	0,07 m	0,27 m	0,33 m
Test 3	(0,55; -0,47)	(-0,59; 0,56)	(0,38; 0,61)	0,47 m	0,45 m	0,41 m

5.2 Implementering av referensmarkörer

För test av referensmarkörer då quadcoptern bars över markörerna kan resultatet med referensmarkörer avläsas i Tabell 5.2 samt resultatet utan referensmarkörer avläsas i Tabell 5.3. Området vilket markörerna var placerade inom var 1x2 m. Markörernas positioner i området var (0,00; 0,80) samt (0,80; 1,75) och referensmarkörernas positioner i området var (0,00; 0,00) samt (0,00; 1,75). Det genomsnittliga felet för alla mätningar med referensmarkörer var 0,11 m, motsvarande fel utan referensmarkörer var 0,30 m.

Tabell 5.2: Mätning av felmarginal på markörpositionering med referensmarkör

	Markör 1	Markör 2	e_1	e_2	$\bar{e}$
Test 1	(0,01; 0,71)	(0,72; 1,83)	0,09 m	0,11 m	0,10 m
Test 2	(-0,01; 0,62)	(0,67; 1,79)	0,19 m	0,19 m	0,19 m
Test 3	(-0,01; 0,74)	(0,78; 1,80)	0,03 m	0,05 m	0,04 m

Tabell 5.3: Mätning av felmarginal på markörpositionering utan referensmarkör

	Markör 1	Markör 2	e_1	e_2	$\bar{e}$
Test 1	(-0,09; 1,08)	(0,93; 2,52)	0,29 m	0,78 m	0,54 m
Test 2	(-0,11; 0,67)	(0,70; 1,68)	0,17 m	0,12 m	0,15 m
Test 3	(-0,02; 0,63)	(0,79; 1,55)	0,17 m	0,20 m	0,19 m

I Tabell 5.4 följer resultaten från återskapandet av test 5.1 med referensmarkörer. Medelfelet för markörerna blev följande: $\bar{e}_1 = 0{,}05$ m, $\bar{e}_2 = 0{,}17$ m och $\bar{e}_3 = 0{,}24$ m vilket kan jämföras med resultaten $\bar{e}_1 = 0{,}32$ m, $\bar{e}_2 = 0{,}40$ m och $\bar{e}_3 = 0{,}47$ m från test 5.1.

Tabell 5.4: Mätning av felmarginal på markörpositionering med hjälp av referensmarkörer, autonomt flygande quadcopter

	Markör 1	Markör 2	Markör 3	e_1	e_2	e_3
Test 1	(0,50; 0,03)	(-0,50; 0,97)	(0,44; 0,72)	0,03 m	0,03 m	0,29 m
Test 2	(0,54; 0,01)	(-0,48; 0,75)	(0,74; 0,88)	0,04 m	0,25 m	0,27 m
Test 3	(0,57; -0,02)	(-0,34; 0,81)	(0,64; 0,96)	0,07 m	0,24 m	0,15 m

5.3 Beräkning av maxhastighet för rörliga markörer

Vid första testet användes den mobila applikation som Parrot tillhandahåller för att se arean bottenkameran registrerade. Vid det andra testet användes istället ROS-paketet tum_ardrone för att se bilden från bottenkameran. Resultaten från båda mätningarna återfinns i Tabell 5.5. Vinklarna θ_x och θ_y beräknades enligt Ekvation 4.1.

Tabell 5.5: Mätning av quadcopterns vinkelupptagning hos bottenkameran.

	Höjd	Δx	Δy	θ_x	θ_y
Test 1	0,80 m	0,75 m	0,36 m	50,2°	25,4°
Test 2	1,50 m	1,46 m	0,83 m	51,9°	30,9°

Om medelvärde på θ_x och θ_y tas erhålls vinklarna $\bar{\theta_x} = 51{,}1°$ och $\bar{\theta_y} = 28{,}2°$. Med hjälp av dessa vinklar beräknades området kameran ser vid 1,5 meters höjd. Enligt

ekvation 4.1 motsvarar detta ett område på $\Delta x = 1{,}25$ m och $\Delta y = 0{,}72$ m. Hastigheten markörer högst får röra sig med blir därför 3,16 m/s enligt ekvation 4.2, förutsatt att quadcopterns position är statisk.

5.4 Detektering av rörliga markörer

Vid testet filmades försöken och en ljudsignal användes för att indikera identifierad markör. Quadcoptern förflyttades från sitt origo, vilket är där quadcoptern befinner sig när systemet initierats, till punkten (0, -0.5) och placerades på en ställning 1,1 m ovanför marken. I Tabell 5.6 nedan följer resultaten från mätningen.

Tabell 5.6: Mätning av felmarginal på markörpositionering av rörlig markör

	Verklig position	Uppskattad position	Fel
Test 1	(0,00; -0,52)	(0,31; -0,18)	0,46 m
Test 2	(0,00; -0,59)	(14.59; -45,18)	46,92 m
Test 3	(0,00; -0,51)	(0,09; -1,22)	0,72 m

Med verklig position menas markörens faktiska position i quadcopterns koordinatsystem vid detektion. Med uppskattad position menas den position som erhölls från systemet vid positionsuppskattning.

5.5 Webbgränssnitt

Panelen som tillfrågades bestod av tio personer. En majoritet, 60%, av dessa är studenter på tekniska utbildningar, de resterande är en blandning av andra studenter och personer i arbetslivet utan akademisk bakgrund. Bild av webbgränssnittet återfinns i appendix som Figur A.1. Betygen från Tabell 5.7 ger ett snittbetyg på 4,3.

Tabell 5.7: Utvärdering av webbgränssnitt

Person	1	2	3	4	5	6	7	8	9	10
Betyg	5	5	4	4	5	4	4	3	4	5

6
Diskussion

Nedan diskuteras de tester som genomförts där resultat samt problem som uppstått analyseras. Vidare diskuteras projektets förbättringspotential samt nya utvecklingsmöjligheter.

6.1 Återskapa masterprojektets resultat med existerande bibliotek

Innan testet kunde genomföras krävdes felsökning av masterprojektets kod då quadcoptern inte flög sin fördefinerade rutt utan endast lyfte och roterade i yaw-led. Felsökningen ledde till identifikation av tidigare ändringar gjorda i tum_ardrone av masterprojektet som inte var dokumenterade. Ändringarna i koden var gjorda för att optimera prestandan för deras quadcopter. På grund av detta användes en ny, omodifierad, version av tum_ardrone. Dock fungerade quadcopterns höjdestimering dåligt och därför återanvändes masterprojektets höjdkorrigering i tum_ardrone.

Då tester för att verifiera första målet genomfördes uppstod problem. Många gånger avvek quadcoptern från förväntat beteende. Ett exempel på detta är att den ibland inte flög den planerade rutten, utan gjorde oväntade manövrar. Orsaken till detta ligger troligtvis i att quadcopterns egna positionsuppskattning var felaktig under dessa flygningar. Detta resulterade i att quadcopterns uppskattade position inte överensstämde med dess faktiska position. Till följd av detta flög quadcoptern fel och nödlandning krävdes för att undvika kollision med utplacerade objekt och väggar. Ett annat tillfälle när quadcoptern hade problem med positionsuppskattningen var vid långa flygningar i y-led. Detta inträffade troligtvis på grund av problem med djupbestämning av PTAMs nyckelpunkter.

Ytterligare ett problem som uppkom var att quadcoptern förlorade nyckelpunkterna som användes för navigering. Då quadcoptern växlar från bottenkameran till framkameran måste quadcoptern lokalisera nyckelpunkterna på nytt, något som quadcoptern ofta misslyckades med. Utöver detta förlorade quadcoptern även nyckelpunkterna när den kom för nära dem. Till följd av detta försökte quadcoptern att återfinna dem och testet fick avbrytas då quadcoptern oftast inte lyckades med detta.

Utöver detta visade det sig att quadcopter, trots höjdkorrigering, sällan flög på den förbestämda höjden. När quadcoptern lyfte flög den till olika höjder vid nästan varje test trots att inga förändringar gjorts mellan testerna. Anledningen till detta beror

med största sannolikhet på positionsuppskattningen av quadcoptern. Efter PTAM-initiering kan quadcopterns uppskattade position ändras, även fast den inte rör sig, om programmet detekterar positionsförändringar hos nyckelpunkterna. Detta kan hända då quadcoptern växlar mellan kamerorna. Höjduppskattningen kan till följd av detta bli lidande och förklara varför quadcopterns flyghöjd är inkonsekvent. Detta problem var något som uppmärksammades under samtliga tester av de olika delmålen.

Vid analys av resultatet visas att den uppmätta felmarginalen ligger runt området 30-60 cm, bortsett från andra mätningen på markör 1. Medelfelet för alla mätningar var 0,42 m, vilket liknar de resultat masterprojektet fick. Anledningen till att resultatet innefattar tre mätningar istället för masterprojektets fem berodde på tidsbrist då flygtiderna vi hade tillgång till var begränsade. Det är även viktigt att poängtera att resultaten från båda mätningarna inte är helt jämförbara i och med det minskade sökområdet. Masterprojektet visade att fel i quadcopterpositionering ackumuleras vid flygning. Detta gör att testen inte utfördes med samma förutsättningar då den totala arean av vårt sökområde var en fjärdedel av masterprojektets totala sökarea. Vi anser på grund av detta att delmål 1.4.1 endast kan räknas som delvis uppnått.

6.2 Implementering av referensmarkörer

I början av projektet var idén att referensmarkörer skulle användas för att kalibrera om quadcopterns position i rummet. Detta genom att quadcoptern korrigerar sin position men behåller PTAM-nyckelpunkterna. Det visade sig inte finnas något färdigt stöd för detta i programmet, utan en egen implementation av detta krävdes. Implementationen krävde djupare kunskap om paketet tum_ardrone vilket saknades. Istället används referensmarkörerna endast till att kompensera för fel i markör- och quadcopterposition innan de laddas upp till databasen. Vid många test visade det sig att ett konstant fel i positioneringen fanns. Detta är tydligt om man kollar på resultaten från både test 1 och 3 i tabell 5.1. Där kan ett konstant fel på ca -0,5 m i y-led utläsas. Detta konstanta fel orsakas av dålig PTAM-prestanda vilket leder till att quadcoptern, vid start, inte uppskattar sin position till origo utan kan variera i x- ,y- och z-led. Då denna implementation ändå klarar av att kompensera för dessa fel samt klarar vårt mål på en förbättring i markörpositionering på 10%, anses lösningen vara tillräckligt bra.

I första testet då quadcoptern endast bars över markörerna kunde en förbättring i markördetektering avläsas samt så kunde den egenskriven koden verifieras. I andra delen av testet kan en markant förbättring i markörpositionering ses. De procentuella förbättringarna i felmarinal för de tre markörerna är 84%, 58% respektive 49% vilket ger en snittförbättring på 64%. Med dessa resultat anses målet vara väl uppnått. Det kan dock tilläggas, att som vid tidigare test var resultat svåra att få då quadcoptern positionsuppskattning fungerade dåligt.

6.3 Detektering av rörliga markörer

Vid testet placerades quadcoptern på en höjd av 1,1 m samt på positionen (0,0; -0,5). Anledningen till höjdvalet berodde på att ställningen som användes begränsade höjden och anledningen till förflyttningen i y-led var på grund av att PTAM-initieringen inte kunde ske där ställningen stod. För att filma testet användes en mobiltelefon och för att generera en ljudsignal skrevs en nod i ROS som genererade ljudsignalen vid detektion av markör. På grund av upplösningen hos kameran blev exakt positionering av markör vid videogranskning svår och kan därmed ha påverkat bestämningen av dess verkliga position med cirka $\pm$ 0,01 m.

Utöver detta var resultat svåra att få. Programmet hade svårt att uppskatta quadcopterns position på grund av dålig PTAM-prestanda och fullständiga resultat kunde inte erhållas från varje test. Test 2 visar detta då systemet identifierat en markör nästan 47 meter från dess egentliga position. Anledningen till denna stora felpositionering var att quadcopterns position under detta test också var grovt feluppskattad av systemet då denna var (14,35; -41,48).

Bortsett från dessa fel visas att quadcoptern har möjlighet att detektera och positionera rörliga markörer. Bestämningen av felmarginalen är dock svår att göra och under testet konstaterades att den valda metoden för bestämning av felmarginal fungerade dåligt. I avsnitt 4.4 diskuteras en alternativ metod för bestämning av felmarginal. Anledningen till att denna inte genomfördes var på grund av tidsbrist i testmiljö.

Delmålet där både quadcoptern och markörerna rör på sig uppfylldes dock inte. Inget test för detta genomfördes på grund av att tid istället lades på implementering av referensmarkörer. Detta på grund av att felmarginalen i markörpositioneringen behövdes förbättras för att få lyckade resultat för positionering av rörliga markörer.

6.4 Webbgränssnitt

Vid utveckling av webbgränssnittet låg fokus på att göra webbgränssnittet användarvänligt. Anledningen till detta var att gruppen valde att se webbgränssnittet endast som ett presentationshjälpmedel, och inte något som används för att faktiskt styra quadcoptern eller felsöka programmet.

Panelen som skulle betygsätta gränssnittet valdes slumpmässigt ut bland vänner och bekanta. Då många studerar på ingejörsutblidningar tillfrågades även personer med annan bakgrund för att ge testet mer tyngd. För alla paneldeltagare förklarades idén med projektet, men inte målet med testet. Med resultatet från panelundersökningen kan det konstateras att målet uppfyllts med god marginal och att webbgränssnittet kan betraktas som mycket tydligt. En större undersökning hade gett ett bättre statistiskt underlag, men då inte huvudmålet med projektet var att utveckla ett webbgränssnitt valdes att inte lägga för mycket tid på undersökningarna.

Utöver att ta in paneldeltagarnas individuella betyg, samlades även konstruktiv kritik och synpunkter in. Kommentarer som exempelvis att det var svårt att förstå vilken enhet de olika axlarna hade samt vad de olika symbolerna i grafen stod för togs emot. Därefter gjordes ändringar i webbgränssnittet baserat på dessa förslag. På grund av det goda resultatet av undersökningen skedde inget ytterligare test för att visa på de nya förbättringarna.

6.5 Förbättringspotential

Då många av projektets problem tros bero på bristfällig PTAM-prestanda tycker vi att quadcopterns positioneringsförmåga är där det finns mest förbättringspotential. Detta problem var även något masterprojektet observerade.

Problemet beror till stor del av att quadcoptern måste växla mellan sina två kameror för att kunna navigera och detektera markörer samtidigt. Varje gång quadcoptern växlar mellan att detektera markörer och navigera måste nyckelpunkterna återigen hittas. Detta betyder även att när quadcoptern letar efter markörer slutar navigeringen att fungera då ingen bildström levereras från framkameran. Utöver detta observerades, de gånger PTAM-prestandan var bra, att positioneringen var bristfällig. Efter initiering av PTAM-nyckelpunkter kunde quadcopterpositionen ändras, trots att den stod still på marken, och även när den inte ändrades var den många gånger feluppskattad.

En utvecklingsmöjlighet skulle därför vara att byta hårdvara så att bildströmmarna från både botten- och framkameran kan skickas och bearbetas samtidigt. På detta vis skulle quadcoptern alltid kunna navigera sig själv samtidigt som den hela tiden letar efter markörer. Detta tror vi skulle bidra till stor förbättring gällande navigeringen i rummet. En annan förbättringsmöjlighet skulle även vara att använda ett annat system för quadcopterpositionering. Då markördetekteringen fungerade bra hade detta kunnat användas för att positionera quadcoptern. I [32] användes en stor ArUco-markör vilket en quadcopter aktivt följde efter. En modifiering av detta arbete, där en statisk markör används för positionsuppskattning istället, skulle kunna användas.

I implementeringen av referensmarkörer skulle en förbättring vara att quadcoptern uppdaterar sin uppskattade position vid detektion av en referensmarkör. I dagens implementation korrigeras endast positionen innan den laddas upp i databasen. Ingen positionskorrigering sker alltså i tum_ardrone, som sköter navigeringen av quadcoptern, och därför påverkas inte den faktiska flygningen.

Ett annat område av projektet som kan förbättras är webbgränsnittet. I dagsläget kan det endast visa hur stort sökområdet är, var quadcoptern befinner sig i området samt var i området quadcoptern har detekterat markörer och då endast dess senaste position och hastighet. Här finns stor förbättringspotential där man exempelvis kan lägga till alternativ för att se tidigare positioner för quadcoptern

och detekterade markörer. Webbgränssnittet skulle även kunna utvecklas för att användas som ett hjälpmedel för felsökning av systemet. En av möjligheterna skulle kunna vara att visa ruttändringen för quadcoptern om förhållanden ändras, till exempel om quadcopterns uppskattade position justeras på grund av detektion av referensmarkör.

7
Slutsats

Under projektet uppkom många problem. Det enda delmål som har uppnåtts helt är utveckling av ett tydligt webbgränssnitt. Resterande delmål har uppnåtts med modifikationer, förutom införandet av rörliga markörer med samma felmarginal som statiska markörer. Det målet har inte uppnåtts då testresultaten visar att felmarginalen vid detektering av rörliga markörer är sämre än felmarginalen för statiska markörer. Syftet med arbetet har inte uppfyllts. Inget test har gjorts för att verifiera om en autonomt flygande quadcopter kan detektera rörliga markörer och räkna ut dess hastighet.

Det kan konstateras att positionsuppskattningen för quadcoptern inte är tillförlitlig och leder till att de olika testresultaten skiljer sig mycket åt. Då positionsuppskattningen för quadcoptern varierar kraftigt mellan varje körning är systemet problematiskt att utveckla då resultatet av en flygning inte går att förutsäga. Med en bättre positionsuppskattning för quadcoptern är även bättre markörpositionering möjlig. Detta har visats då implementeringen av referensmarkörer har lett till en förbättring av felmarginalen för markörpositioneringen.

På grund av att positionsuppskattning av quadcoptern inte anses tillförlitlig är det svårt att få tillförlitliga resultat. Det krävs många försök för att få några resultat på de tester där quadcoptern är inblandad, men då resultat faktiskt erhålls är de ofta inom samma felmarginal.

Vi anser att projektet i sin helhet inte är lyckat, dock kan vi dra några viktiga slutsatser. För vidareutveckling av systemet behövs en bättre positionsuppskattning för quadcoptern. Med en bättre positionsuppskattning anses det att målet om detektion och hastighetsuppskattning av rörliga markörer hade kunnat uppfyllas. Utöver detta anses webbgränsnittet vara välutvecklat och färdigt att användas i andra liknande projekt.

Litteraturförteckning

[1] K. Nonami, *Prospect and Recent Research & Development for Civil Use Autonomous Unmanned Aircraft as UAV and MAV*. Department of Electronic and Mechanical Engineering, Chiba University, Japan, 2007.

[2] "Smarta användningsområden för drönare", *Fastighetsmäklarförbundet*, 2015; `http://fmf.se/smarta-anvandningsomraden-for-dronare/` (Hämtad: 28 jan., 2016)

[3] K. Eriksson, "Drönare överallt - så här används de obemannade farkosterna idag", *M3*, 2014; `http://m3.idg.se/2.1022/1.593056/dronare-overallt---sa-har-anvands-de-obemannade-farkosterna-idag` (Hämtad: 28 jan., 2016)

[4] A. Josefsson, A. Johansson, E. Rosenberg, J. Benjaminsson, K. Svensson, "KandiDrone - ett steg närmare autonom flygning", kandidatuppsats, Institutuionen för signaler och system, Chalmers tekniska högskola, 2015.

[5] "Drönare", *Transportstyrelsen*, `http://www.transportstyrelsen.se/dronare` (Hämtad: 3 feb., 2016)

[6] R. Roophnath, S. Maxwell, *Miniature Wireless Quadcopter* Department of Electrical Engineering, University of Hartford, USA, 2014.

[7] M. Anwar Ma'sum, Grafika Jati, M. Kholid Arrofi, Adi Wibowo, Petrus Mursanto, Wisnu Jatmiko, *Autonomous Quadcopter Swarm Robots For Object Localization and Tracking* Faculty of Computer Science Universitas Indonesia, Indonesien.

[8] Arjun M. Nair, Manoj A.Talreja, Prithvi Bhaskar, *Design Solution for flight Control of A Quadcopter* Department of Telecommunication, Dayananda Sagar College of Engineering, Indien, 2014.

[9] F. Arifin, R. A. Daniel, D. Widiyanto, *Autonomous Detection and Tracking of an Object Autonomously Using AR.DRONE Quadcopter* Faculty of Computer Science, Universitas Indonesia/ Faculty of Computer Science, UPN Veteran, Indonesien.

[10] M. Lindelöw, H. Olsson, D. Widmann, L. Wikander, "Marker Detection and Autonomous Flight Using Quadcopter Mounted Cameras", designprojekt, Institutionen för signaler och system, Chalmers tekniska högskola, Sverige, 2015.

[11] "Detection of ArUco Markers", *OpenCV*, 2015; `http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html#gsc.tab=0` (Hämtad: 6 apr., 2016)

[12] S. Garrido-Jurado, R. Muñoz-Salinas, F.J Madrid-Cuevas, M.J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under

occlusion", Department of Computing and Numerical Analysis, University of Córdoba, Spanien.

[13] J. Engel, J. Sturm, D. Cremers, "Scale-Aware Navigation of a Low-Cost Quadrocopter with a Monocular Camera", Department of Computer Science, Technical University of Munich, Tyskland.

[14] "AR.Drone 2.0: Technical Specifications", *Parrot* http://ardrone2.parrot.com (Hämtad: 11 mar., 2016)

[15] S. Riisgard, M. Rufus Blas, "SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping" http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf (Hämtad: 23 feb., 2016)

[16] "Source Code", *Parallel Tracking and Mapping for Small AR Workspaces*, 2014; http://www.robots.ox.ac.uk/~gk/PTAM/ (Hämtad: 23 feb., 2016)

[17] "About ROS", *ROS*, http://www.ros.org/about-ros/ (Hämtad: 3 feb., 2016)

[18] J.M. O'Kane, "A Gentle Introduction to ROS", 2014; https://cse.sc.edu/~jokane/agitr/agitr-small-service.pdf

[19] D. Hemmendinger, "HTML" i *Encyclopædia Britannica*. (Online); http://academic.eb.com/EBchecked/topic/279729/HTML (Hämtad: 22 mar., 2016).

[20] W. Harrel, *HTML, CSS, and JavaScript Mobile Development For Dummies*, John Wiley & Sons, 2011, Kapitel 2: Understanding CSS. (E-bok) Tillgänglig: http://library.books24x7.com.proxy.lib.chalmers.se/assetviewer.aspx?bookid=43220&chunkid=600444387&rowid=81.

[21] D. Cameron, *HTML5, JavaScript and jQuery 24-Hour Trainer*, Wrox Press, 2015, Kapitel 11: JavaScript. (E-bok) Tillgänglig: http://library.books24x7.com.proxy.lib.chalmers.se/toc.aspx?bookid=82361

[22] "Learn More", *JavaScript.com*, 2015; https://www.javascript.com (Hämtad: 11 mar., 2016)

[23] N. Gehani, *The Database Application Book*, Silicon Press, 2011. (E-bok) Tillgänglig: http://library.books24x7.com.proxy.lib.chalmers.se/toc.aspx?bookid=42026

[24] "What is PHP?", *PHP*, 2016; http://php.net/manual/en/intro-whatis.php (Hämtad: 2 apr., 2016)

[25] "Introducing JSON", *JSON*; http://json.org (Hämtad: 11 mar., 2016)

[26] "OpenCV", *OpenCV*, 2016; http://opencv.org/ (Hämtad: 21 feb., 2016)

[27] "vision_opencv", *ROS*, 2015; http://wiki.ros.org/vision_opencv (Hämtad: 6 apr., 2016)

[28] "aruco", *ROS*, 2015; http://wiki.ros.org/aruco (Hämtad: 21 feb., 2016)

[29] "ardrone_autonomy", *ROS*, 2015; http://wiki.ros.org/ardrone_autonomy (Hämtad: 21 feb., 2016)

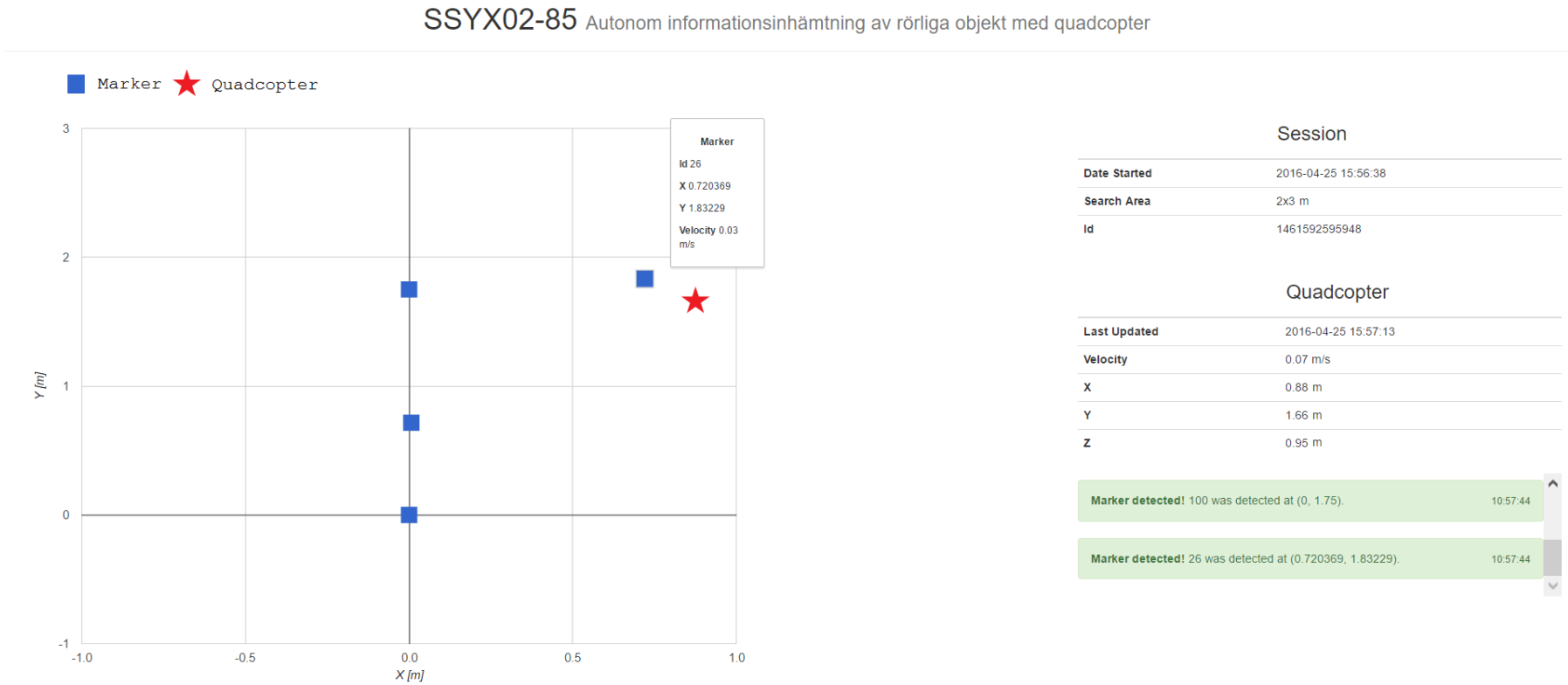
[30] "Sending Commands to AR-Drone", *ardrone_autonomy indigo-devel documentation*, http://ardrone-autonomy.readthedocs.org/en/latest/commands.html(Hämtad: 6 apr., 2016)

[31] "tum_ardrone", *ROS*, 2015; http://wiki.ros.org/tum_ardrone(Hämtad: 23 mar., 2016)

[32] B. Nilsson, T. Nilsson, G. Snorri Ragnarsson, M. Tulldahl, ”Positioning of a Quadcopter Relative to a Visually Tracked Marker”, designprojekt, Institutuionen för signaler och system, Chalmers tekniska högskola, 2015.

A

Figurer



Figur A.1: Bild på webbgränssnitt.