CODEN:LUTEDX/(TEIE-3126)/1-62/(2023)

Industrial Electrical Engineering and Automation

Utvärdering av systemprestanda och tillförlitlighet genom en simulerad operativ miljö med en poängbaserad modell

Jesper Strand
Benjamin Nilsson Lundqvist

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Utvärdering av systemprestanda och tillförlitlighet genom en simulerad operativ miljö med en poängbaserad modell

Kandidatuppsats

LUND
UNIVERSITY

Jesper Strand
Benjamin Nilsson Lundqvist

Avdelningen för industriell
elektroteknik och automation

Handledare: Christian Nyberg
Examinator: Mats Lilja

12 Juni 2023

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Lunds universitet
Lund 2023

Sammanfattning

Examensarbetet fokuserar på utvärdering samt testning av FLEXNET, ett Unattended Ground Sensor (UGS) system utvecklat av Bertin Exensor. FLEXNET:s grundläggande syfte är att övervaka stora områden utan att kräva omfattande resurser i form av personal som aktivt övervakar och underhåller systemet på plats.

Vid utvecklandet av större system kan det ofta vara besvärligt att utföra fullskaliga tester av hela systemet i företagets egna lokaler. I dagsläget har företaget vid mer omfattande test, tvingats rigga upp hela systemet ute i fält. Detta har visat sig vara mycket resurskrävande samt att omgivningens parametrar kan variera från dag till dag. Därför undersöker och implementerar detta examensarbete i samråd med företaget, möjligheten att testa ett komplett system med hjälp av en Raspberry Pi och Python för att simulera en utomhusmiljö för systemet i företagets lokaler. Målet med examensarbetet är att utveckla en pålitlig och omfattande testrigg som underlättar för de anställda att fortlöpande testa sina produkter utan att behöva lämna kontoret.

Utvecklingsprocessen genomfördes i flera etapper. Inledningsvis utforskade examensarbetarna metoder för att fysiskt ansluta till enheterna. Därefter genereras sensordata till sensorerna för att utvärdera om det resulterar i en adekvat larmsignal. Slutligen skickades data till flera sensorer simultant för att analysera det övergripande systemets interaktioner. Resultaten från examensarbetet indikerar att det är möjligt att använda en Raspberry Pi för att simulera en utomhusmiljö för systemet inom kontorsmiljön. Dessutom erhålls värdefull information om systemets prestanda med hjälp av statistisk modellering.

Keywords: Systemtestning, Seriell kommunikation, Python, Elektronik, Automation

Abstract

The thesis focuses on the evaluation and testing of FLEXNET, an Unattended Ground Sensor (UGS) system developed by Bertin Exensor. The primary purpose of FLEXNET is to monitor large areas without requiring extensive resources in the form of personnel actively monitoring and maintaining the system on site.

In the development of comprehensive systems it can often be challenging to carry out full-scale tests of the entire system within the confines of a company's own premises. Currently for more extensive tests the company has been forced to set up the entire system in the field. This has proven to be very resource-intensive, and the parameters of the environment can vary from day to day. Therefore, this thesis, in consultation with the company, explores and implements the possibility of testing a complete system using a Raspberry Pi and Python to simulate an operational environment for the system inside the office. The goal of the thesis is to develop a reliable and comprehensive test rig that makes it easier for employees to continuously test their products without having to leave the office.

The development process was carried out in several stages. Initially, the thesis workers explored methods for physically connecting to the devices. Subsequently, sensor data was generated for the sensors to evaluate whether it resulted in an adequate alarm signal. Finally, several sensors were fed data simultaneously to analyze the overall system's interactions. The results from the thesis indicate that it is possible to use a Raspberry Pi to simulate an operational environment for the system within the office setting. Additionally, it provides and scores valuable information about the system's performance using statistical modeling.

Keywords: System Testing, Serial Communication, Python, Electronics, Automation

Förord

Detta examensarbete representerar slutpunkten för våra studier på LTH, och vi är både stolta och tacksamma över att kunna presentera vårt arbete. Vi vill börja med att uttrycka vår tacksamhet till alla som har hjälpt oss under denna resa

Först och främst vill vi rikta ett innerligt tack till vår handledare, Ulf Hörndahl. Hans kontinuerliga stöd och rådgivning har varit helt ovärderliga för oss under hela denna process. Hans beredskap att ge råd, ställa utmanande frågor och hjälpa oss navigera i den komplexa värld som detta examensarbete har fört oss in i har varit en starkt bidragande faktor till arbetets framgång.

Vi vill framföra vårt djupa tack till alla på Bertin Exensor, särskilt serviceteamet, som har välkomnat oss med öppna armar. Ert engagemang, kunskap och generositet har varit av avgörande betydelse för oss.

Slutligen vill vi tacka Christian Nyberg för hans handledning under examensarbetets gång, speciellt för all bra och värdefull feedback vi fått på rapporten och planeringen. Utöver det vill vi tacka vår examinator Mats Lilja för att han granskat rapporten.

Innehåll

KAPITEL 1

Introduktion

Detta inledande kapitel syftar till att introducera läsaren till syftet och bakgrunden till det aktuella examensarbetet. Vidare behandlar det även mål- och problemformuleringar samt avgränsningar för att säkerställa att arbetet är genomförbart.

1.1 Bakgrund

Bertin Exensor är ett företag som utvecklar Unattended Ground Sensor system (UGS), dessa system marknadsförs i första hand till försvarindustrin, men säljs även till privata kunder. Ett system kan byggas upp på många olika sätt men traditionellt består det av en basstation vars uppgift är att sköta kommunikationen mellan sensorenheterna och lagra datan som sedan kan hämtas ut och presenteras via ett Human Machine Interface (HMI). Olika sensorer med olika tröskelvärden detekterar och genererar larm vid överskridande av specifika gränsvärden. Detta beskriv mer detaljerat i kapitel 2.5

I dagsläget saknas lösning för att testa systemet i sin helhet i företagets lokaler. Lösningar har varit att företaget, vid mer omfattande test, tvingats rigga upp hela systemet ute i fält. Denna tidskrävande process leder till färre tester av systemet då det kräver en personalinsats som anses vara oförsvarbar. Detta resulterar i en fördröjd feedback till utvecklingsenheten och problem upptäcks sent i produktcykeln.

Ett förslag på en lösning är att använda en Raspberry Pi och simulera en utomhusmiljö för systemet på kontoret. Alltså generera larm på sensorerna och ersätta vad kameralinsen egentligen ser och mata den med testbilder. Dessa testfall upprepas sedan för olika inställningar på kamerakroppen och sensorerna samt för olika radioprotokoll och inställningar. Detta kommer då ge företaget data på vilka inställningar som påverkar prestandan i systemet.

Utvecklingen kommer ske i följande ordning. Först kommer examensarbetarna undersöka hur testriggen ska koppla in sig på enheterna fysiskt. Därefter kommer testrig-

gen mata sensorerna med befintlig sensordata och undersöka om utfallet blir önskad larmsignal. Slutligen kommer arbetet innebära att mata flera sensorer simultant för att undersöka hela systemets samspel.

1.2 Syfte

Syftet är att testa kameror och utrustning genom inspelad eller genererad sensordata. Detta för att löpande kunna testa utrustningen utan att avsätta personal samt för att undgå behovet att alltid tvingas testa kameror ute i fält. Resultatet ger även en bra jämförelse av prestanda utifrån valda inställningar. Detta kommer underlätta utvecklingen genom att automatiskt hitta bästa inställningar. Utöver det ger det även ett mått på prestanda eller detektionsförmåga av produkterna

1.3 Målformulering

Examensarbetet ämnar utforska huruvida sensorer och utrustning svarar enligt förväntningarna, samt dokumentera frekvensen av reproducerbara resultat vid given indata till systemet. Om möjligt bör kameran provas i kombination med sensorer inomhus i en testmiljö som ska bidra med insikter som vägleder vid framtida utveckling.

1.4 Problemformulering

Examensarbetet kommer att svara på dessa frågor:

1. Är det möjligt att använda existerande mjukvaror för att testa systemet?
2. Hur återspelar applikationen på bästa sätt inspelad data till sensorerna?
3. Hur skickar applikationen informationen via företagets AUX-interface?
4. Hur presenteras statistik som på ett informativt sätt beskriver antal lyckade testsekvenser?
5. (a) Hur poängsätts ett resultat i ett testsystem som inte bara har två utfall?
   (b) Vad är en lämplig skala för poängsättningen?
   (c) Vilka parametrar poängsätts systemet utifrån?
6. (a) Är en Raspberry Pi kraftfull nog för att kunna simulera hela systemet?
   (b) Är en Raspberry Pi snabb nog för att kunna beräkna resultatet av testfallen under simulering?
7. Hur hanterar applikationen när flera enheter vill kommunicera samtidigt?

1.5 Motivering

Examensarbetarna anser att detta arbete kommer behandla flertalet av de ämnesområden som har studerats under utbildningen. Problemet i fråga har en intressant bredd då den innefattar delar från både grundläggande analog signalhantering till mjukvaruutveckling i ett högnivåspråk. För företaget hoppas examensarbetarna kunna utveckla en pålitlig och omfattande testrigg som hjälper de anställda att löpande testa sina produkter utan att behöva lämna kontoret.

1.6 Avgränsningar

För att upprätthålla tidsplanen förhåller sig arbetet till följande avgränsningar:

- Applikationen ska inte generera ny verklighetstrogen data. Det som spelas upp är förinspelad data.
- Applikationen ska inte skapa kommunikationen mellan enheter, utan endast testa systemet som är på plats.
- Applikationen ska inte analysera innehållet i bilden från kameran, utan endast analysera om kameran har tagit en bild eller inte och i så fall vid vilken tidpunkt.
- Testriggen eller applikationen behöver inte klara sig längre än 48 timmar utan mänsklig input.

KAPITEL 2

Teknisk bakgrund

Detta kapitel behandlar de tekniker och verktyg inom programmering och elektroteknik som har används under examensarbetets gång. Kapitlet fokuserar på specifika Python-moduler som används för datahantering samt grundläggande tekniker för seriell kommunikation.

2.1 Python

Python är ett objektorienterat programmeringsspråk som utmärker sig genom sin tydliga syntax och kraftfulla funktionalitet [1]. Python är kompatibelt med flertalet plattformar, såsom Unix-varianter, Linux, macOS och Windows. Python kan även integreras och utökas med C/C++ vilket är en fördel för examensarbetet då dessa språk förekommer i stor omfattning på företaget.

2.1.1 Time

För att hantera tider i moduler, användes *time*-modulen [2]. Den har en metod *.sleep()* som stannar upp exekveringen av programmet i valt antal sekunder. Samt en metod *.time()* som returnerar systemtiden i sekunder från epoch formaterat som en float. Epoch är en referenspunkt i tiden, vanligtvis 00:00:00 UTC den 1 januari 1970, från vilken tidsintervall och datum mäts inom datorsystem.

2.1.2 JSON

JSON (JavaScript Object Notation) är ett lättviktigt datautbytesformat som är lätt att läsa och skriva för människor, samtidigt som det är lätt att tolka och generera för datorer [3]. JSON är inte kopplat till ett specifikt språk och kan användas i flertalet tillämpningar. Examensarbetarnas implementerade JSON för att möjliggöra kommunikation med företagets basstation genom att använda ett färdigutvecklat API, vilket står för Application Programming Interface och utgör en uppsättning regler och

protokoll för att tillåta olika mjukvarusystem att kommunicera och interagera med varandra.

Projektets hantering av JSON skedde med hjälp av modulen *json* [4]. Den tar in Pythons datatyper och gör om dem till JSON:s egna format med hjälp av metoden *.dumps()*. Omvänt går att göra med metoden *.loads()*. Med hjälp av *.loads()* är det möjligt att göra om en JSON-sträng till Python för att ändra värden och sedan spara ner den nya versionen av JSON-strängen med hjälp av *.dumps()*.

2.1.3 Matplotlib

Matplotlib är ett omfattande visualiseringsbibliotek för Python, som används för att skapa 2D-grafer och diagram [5]. Biblioteket erbjuder en mängd olika plottyper och grafstilar, inklusive linjediagram, stapeldiagram, errorbardiagram och histogram.

2.1.4 Numpy

NumPy (Numerical Python) är ett paket för numeriska beräkningar i Python [6]. Det erbjuder stöd för stora, multidimensionella arrayer och matriser samt en omfattande samling av matematiska funktioner för att utföra effektiva operationer på dessa datastrukturer. *NumPy* används ofta inom vetenskaplig och teknisk forskning för att underlätta hantering av stora datamängder och genomföra komplexa matematiska beräkningar.

2.1.5 Statistics

Statistics-modulen är en del av Pythons standardbibliotek och ger grundläggande statistiska funktioner för att analysera och sammanfatta data [7]. Modulen inkluderar funktioner för att beräkna medelvärde, median, varians, standardavvikelse och kvantiler, samt fördelningsrelaterade funktioner såsom normalfördelning och Poissonfördelning.

2.2 Seriell kommunikation

Seriell kommunikation är en teknik som implementeras inom både elektronik och mjukvara, där data överförs sekvensiellt bit för bit över en enskild kommunkationskanal [8].

2.2.1 UART

UART (Universal Asynchronous Receiver/Transmitter) är ett seriekommunikationsprotokoll som tillåter data att överföras asynkront, utan en separat klocksignal. För att använda UART-protokollet behöver avsändaren och mottagaren komma överens

om en gemensam baudrate, det vill säga bestämma hastigheten i vilket datan kommer överföras.

En ram i UART-protokollet består av en startbit, en uppsättning databitar, en paritetsbit och en eller flera stoppbitar [8]. Startbiten markerar börjar av databitarna. Paritetsbitar används för att kontrollera att meddelandet är felfritt genom att summera 5-9 bitar av datan. Om summan är jämn så är hela paketet korrekt. Databitar innehåller den information som ska överföras och stoppbitar indikerar slutet av varje paket. Paritet, data och stoppbitarna kan konfigureras för att uppfylla specifika kommunikationskrav.

Baudrate är en avgörande del i UART-kommunikationen eftersom den bestämmer hastigheten som datan förväntas att skickas och tas emot. Det är nödvändigt att båda enheterna har samma baudrate-inställningar för att säkerställa korrekt kommunikation mellan dem. Vanliga baudrate-hastigheter inkluderar 9600, 19200, 38400, 57600 och 115200 baud, men andra hastigheter kan också användas beroende på tillämpningen.

2.3 Raspberry Pi

Raspberry Pi (RPI) är en enkortsdator utvecklad av det brittiska företaget Raspberry Pi Foundation [9]. Datorn har blivit populär i utbildning, prototyputveckling och olika IoT-applikationer på grund av sin lilla storlek och användarvänlighet.

Raspberry Pi-arkitekturen utgörs av en ARM-baserad processor, vilken är förekommande i ett flertal mobiltelefoner. Utöver processorn finner man andra komponenter, däribland RAM-minne, flashminne, en HDMI-anslutning, USB-portar, TRS-jack, ethernet-port samt general-purpose input/output(GPIO) stift. Datorn drivs av ett Linuxbaserat operativsystem vid namn Raspbian OS. Då operativsystemet är en distribution av Linux får examensarbetarna även tillgång till SSH. Secure Shell (SSH) är ett krypterat nätverksprotokoll som används för att via internet kunna styra enheten.

Utöver de standardkomponenter som finns på Raspberry Pi kan användaren lägga till en stor mängd olika tillbehör och sensorer för att ytterligare utöka enhetens funktionalitet och anpassningsbarhet för programmering i både hög- och lågnivå språk. Exempel på dessa tillbehör inkluderar kameramoduler, skärmar, olika typer av sensorer och annan hårdvara.

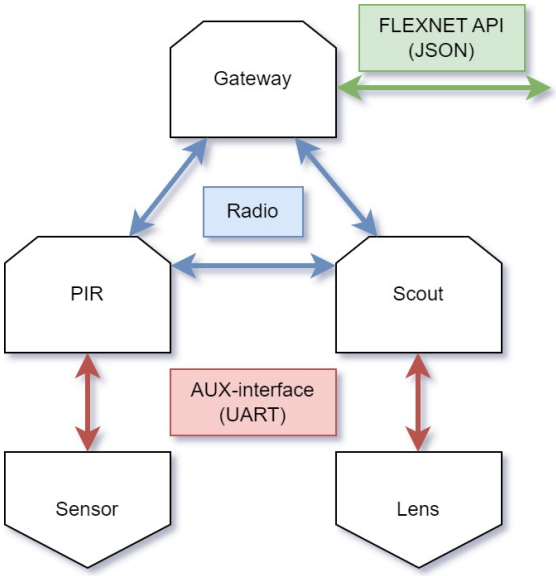
2.4 Analytic Hierarchy Process

Analytic Hierarchy Process (AHP) är en teknik för att hantera komplexa beslutsproblem [10]. Den utvecklades av *Dr. Thomas L. Saaty* på 1970-talet och har en bred tillämpning som innefattar bland annat planering, resusallokering samt prioritering av projekt. AHP hjälper beslutsfattare att organisera och rangordna sina krav genom att dela upp i olika alternativ. Detta görs genom att bryta ner problemet i mindre beståndsdelar och värdesätta vilken tyngd dessa alternativ har för proble-

mets lösning. Varje alternativ kallas kriterier och kan ha egna delkriterier. Parvisa jämförelser används för att bestämma betydelsen av varje kriterium i förhållande till varandra. Resultatet blir en uppsättning viktade poäng för varje kriterium, vilket leder till möjligheten att rangordna dem efter lämplighet.

2.5 FLEXNET wireless surveillance platform

Systemet som examensarbetarna har utvärderat heter FLEXNET vilket är ett Unattended Ground Sensor (UGS) system [11]. Systemets grundläggande funktion är att kunna övervaka stora områden utan att behöva omfattande resurser i form av personal som aktivt är på plats och sköter om systemet. Systemet agerar som ett meshnätverk där Gateway fungerar som en basstation och sensorerna fungerar som självständiga noder.



Figur 2.1: Illustration över hur enheterna kommunicerar med varandra samt var UART- och JSON-kommunikationen sker i systemet.

Det existerar en mängd olika möjliga systemkonfigurationer och dess komplexitet ökar i takt med systemets storlek och omfattning. För att följa examensarbetets begränsningar och ramar, beslutades det att använda en tillämpning som består av en Gateway, en PIR-sensor och en Scout Mk3-kamera. Varje sensor fungerar som en individuell nod i Mesh-nätverket, och har förmåga att vidarebefodra data från de andra sensorerna till basstationen. Systemet som testades illustreras i figur 2.1

2.5.1 Gateway

Gateway i figur 2.2 är den centrala enheten där all information samlas och lagras i FLEXNET-system [12]. Gateway implementeras antingen som en basstation eller som en repeater som rapporterar till basstationen. Systemet kräver ingen uppkoppling och

kan fungera obemannat i månader. Det är utifrån denna basstation som examensarbetarna hämtar, hanterar och validerar om en larmkedja har utlösts. Här hämtar examensarbetarna även informationen som i ett senare skede analyserar hur lång tid det tog från första larmet tills en bild togs och basstationen har registrerat larmet.

Figur 2.2: En bild på en Gateway.

2.5.2 PIR

PIR-en i figur 2.3 är en passiv infraröd sensor som läser av den emitterade infraröda strålningen från förbipasserande föremål [13]. PIR-en kan även avgöra åt vilket håll som föremålet passerar framför sensorn. När PIR-en har noterat en rörelse så skickas denna informationen vidare. I examensarbetarnas tillämpning så skickas larmsignalen till nästa sensor vilket är Scout Mk3 som i sin tur börjar med sin algoritm.

En PIR består av en basenhet med integrerad radio, GPS-mottagare och batteri till vilken det kopplas ett utbytbart PIR-rör. Det finns tre varianter av PIR-röret: kort, medium och lång räckvidd, med olika detektionsområden och synfält. PIR-röret kan anslutas som en extern komponent till vilken annan typ av FLEXNET-sensor som helst.

2.5.3 Scout Mk3

Scout Mk3 är en enhet som består av en kamerakropp och ett, eller flera, kamerahuvuden. I systemet finns flera olika kamerahuvuden [14]. De kan levereras med termiska (TI) sensorer och ljuskänsliga sensorer med olika linsalternativ, såsom zoom eller smal-/vidvinklad. Kamerahuvuden kan antingen monteras direkt på kamerahuset eller monteras på avstånd och anslutas via en kabel. I examensarbetarnas fall så aktiveras Scout Mk3 i figur 2.4 av PIR-ens larm över radionätverket.

Figur 2.3: En bild på PIR med tillhörande PIR-rör.

Figur 2.4: En bild på en Scout med ett kamerahuvud med två kameramoduler (IR och svartvit).

Efter aktivering startar rörelsedetektion i en förinställd tid konfigurerad av operatören. Även Scout Mk3 fungerar som en individuell nod i mesh-nätverket och har förmåga att vidarebefordra data från andra sensorer tillbaka till basstationen.

KAPITEL 3

Metod och Analys

För att effektivt hantera ett omfattande arbete med flera olika delmoment, valde examensarbetarna att strukturera arbetet i olika faser. Det beslutades att initialt enbart utveckla enkla test samt informationsökning. När examensarbetarna blev bekväma med företagets produkter samt utvecklingsmiljön för Raspberry Pi, var det dags att fokusera på projektspecifika uppgifter. Under den första fasen lades stor vikt på kodstrukturen samt examensarbetes utformning. Utöver detta ägnades cirka en tredjedel av tiden i fas ett åt att utforska möjligheterna med de resurser examensarbetarna förfogade över. Examensarbetarna estimerade även vilken kunskap som realistiskt skulle förvärvas under examensarbetet och vad som var möjligt att köpa in.

Gemensamt för faserna var att examensarbetarna tog med kunskap och kod från föregående fas in till nästa fas. Ny kod testades separat i sin egen modul. Omskrivning skedde i samband med övergång mellan faser för implementering med resterande system. Därefter integrerade nästkommande fas nya funktioner som till exempel en ny enhet, nya moduler eller öka tillförlitligheten på nuvarande kod.

3.1 Utvecklingsfas ett

Under denna fas etablerades grundstrukturen för examensarbetet, inklusive val av programmeringsspråk samt hårdvara. Det var även under denna period som majoriteten av informationsinsamlingen genomfördes. Vidare började en skiss över kodstrukturen att ta form. Under samma fas utvecklades även UART- och JSON-paketen för att inleda utvecklingen av ett grundläggande testfall.

3.1.1 Informationsökning

Efter det inledande mötet med företaget påbörjades planeringsfasen för examensarbetet. Genom diskussioner och övervägande i samråd med företaget, fastställdes det att en Raspberry Pi vara lämplig för den planerade implementationen.

Raspberry Pi visade sig vara ett lämpligt val för detta examensarbete av flertalet anledningar. Systemet är mycket populärt och har en stor användarbas samt omfattande dokumentation. Detta underlättade för utvecklarna som ville komma igång snabbt då mycket kunskap fanns att hämta online.

Denna enkortsdator är utrustad med ett flertal in- och utgångar som ger möjlighet till inkoppling av tangentbord, mus och video vilket underlättade användningen. Dessutom finns GPIO pinnar som examensarbetarna använde för seriell kommunikation med olika elektroniska komponenter. Vidare på grund av att kortet var uppbyggt som en dator, underlättades kommunikationen med de andra enheterna. Raspberry Pi:s lämplighet visade sig också när det kommer till programmeringsspråk, eftersom Raspberry Pi var väl kompatibelt med både Python och C++. Detta gav examensarbetarna möjlighet att använda båda språken om så skulle krävas. Detta märktes, och var enligt Eben Upton, en av grundarna av Raspberry Pi, anledningen till ordet *Pi*. Han berättade i en intervju:

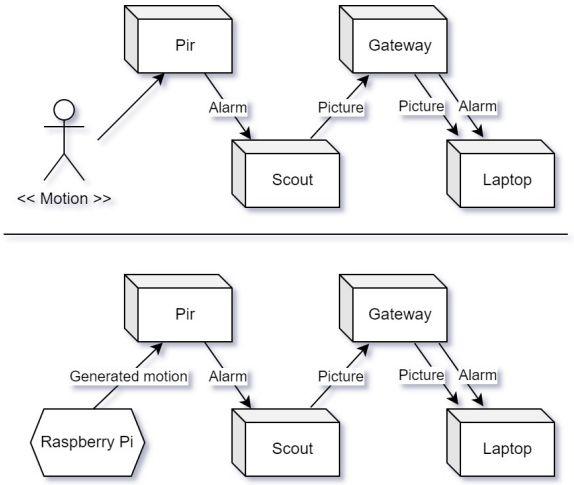
"Pi is because originally we were going to produce a computer that could only really run Python. So the Pi in there is for Python." [15]

Det fanns flera olika modeller av Raspberry Pi och då examensarbetet kräver omfattande processorkraft, beslutades det att använda den senaste modellen. Detta var en 4B som har fyra gånger mer ram än på modell 3B. Ytterligare fördelar med en 4:a var att den hade två USB3 samt två videoutgångar vilket eventuellt skulle behövas. Nackdelen med en 4B var att den kostade 282kr mer hos den rekommenderade återförsäljaren i Sverige. Det är även så att 4:an behövde mer strömförsörning för att uppnå den högre prestandan. Efter att ha övervägt fördelar mot nackdelar valdes 4:an. Den enda anledningen att använda en 3:a var att den redan skulle funnits tillgängligt utan att behöva köpas in. Se [16][17][18]

Valet av programmeringsspråk föll på Python på grund av att det uppfattades som enklast för examensarbetet, samt den goda funktionaliteten med Raspberry Pi som var den valda plattformen. Arbetet hade kunnat skrivas i C++ men efter att ha tittat på introduktioner till båda språken framstod Python som mest lämplig då den liknar den kunskap examensarbetarna redan besatt. Förutom detta så var majoriteten av kodexempel och information som fanns tillgängligt skrivet utifrån att Python används på Raspberry Pi, speciellt när det gäller kommunikation via GPIO-stiften. Slutligen var Python ett kraftfullt språk med ett mycket omfattande bibliotek som innehåller en mängd hjälpfunktioner. Vilket ledde till att examensarbetarna ansåg det vara värt att investera tid i att förvärva denna kunskap, eftersom det även skulle vara fördelaktigt för kommande projekt.

Introduktion på företaget

Efter det att examensarbetarna introducerats till testavdelningen inleddes en undersökning med personalen på avdelningen för att förstå hur systemet fungerar. Under denna process erhöll examensarbetarna en klar insikt om vilka delar av systemet som var viktiga att utvärdera. Dessa delar visualiseras i figur 3.1.



Figur 3.1: Flödesschema för hur system fungerar samt hur en Raspberry Pi ersätter en riktig rörelse.

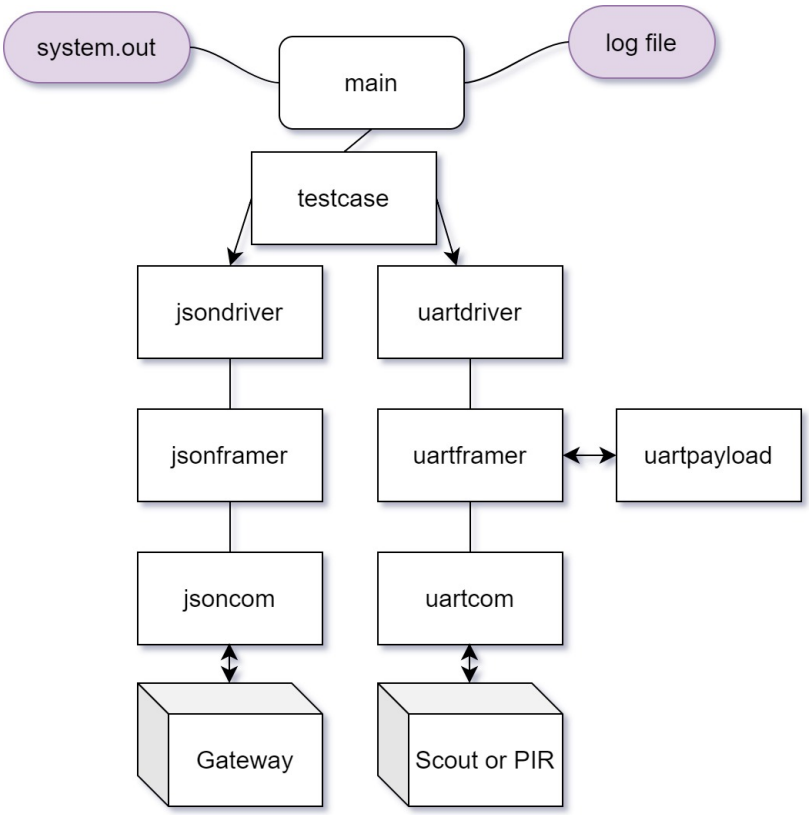
Examensarbetarna började med att undersöka PIR-en genom en omfattande genomgång av befintlig dokumentation. Utifrån denna dokumentation stod det klart att examensarbetet även behöver undersöka kommunikationsprotokollet UART, vars uppgift i examensarbetet var att hantera dataöverföring mellan Raspberry Pi och de övriga enheterna. Slutligen hade examensarbetarna tillräckligt med kunskap för att börja arbetet med att återskapa den funktionalitet som PIR-en hade samt att kunna generera larm. När examensarbetarna säkerställt att larmgenereringen fungerande som förväntat, fortsatte övriga funktioner av PIR-en att implementeras.

Därefter inleddes arbetet med Scout Mk3. Första delen av implementationen var att återskapa de protokoll samt funktioner som var unika för Scout-en. Dessa protokoll var snarlika de som återfanns i PIR-en och lärdomarna från föregående implementation kunde appliceras. Andra delen av implementationen var att ersätta kamerahuvudet med examensarbetarnas Raspberry Pi för att kunna styra vad som spelades upp för Scout-en.

3.1.2 Kodstruktur

Efter planeringen och informationssökning var klar bestämdes också kodstrukturen som illustreras i figur 3.2. Kodstrukturen utformades med ett mönster, inte bara för att underlätta programmeringsprocessen, utan även för att förbättra förståelsen och läsbarheten av koden. Strukturen byggdes på en huvudsmodul *main*, som styrde de mest abstrakta delarna i koden. Den såg till att användaren hade konstant feedback i konsolen och att *testcase* kördes och loggades vid sidan om. *Main* begär nästa testfall och sen var det upp till koden i *testcase* att generera samt utföra testet. Gemensamt för *main* och resten av modulerna var att de använder biblioteket *time* och dess metoder *.sleep()* eller *.time()* för att hantera timings och tidsberäkning. Alla moduler importerar även en konfigurationsfil som innehåller alla konstanter. Detta för att separera ut allt som användaren ska kunna ändra i en separat fil.

I *testcase.py* fanns två moduler *uartdriver* och *jsondriver* vars uppdrag var att ta hand

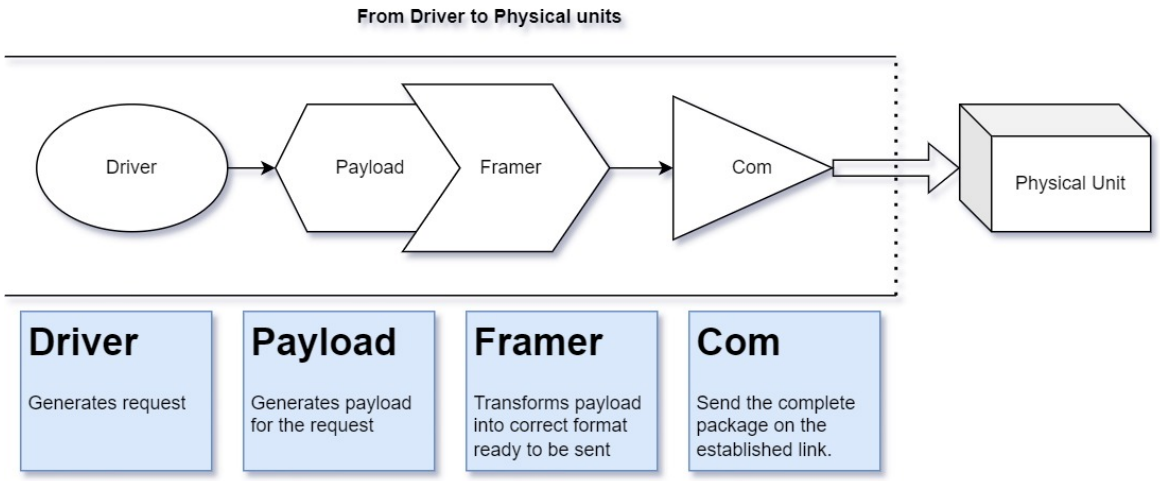


Figur 3.2: Kodstrukturen i första fasen. Figuren visar hur modulerna som utvecklats samverkar, samt hur de var kopplade till de fysiska enheterna och logfilen.

om anslutningar till fysiska enheter. De hade båda egna moduler som skötte kommunikationen, encoding och decoding. Men drivernas uppgifter var att vara knutpunkten för dessa moduler och hålla all logik och regler. För att upprätthålla förbindelsen parallellt med alla enheter kördes varje driver på en egen tråd. Ifall ingen information kom till drivern från *testcase* skulle den utföra minsta möjliga för att upprätthålla förbindelsen. När sedan *testcase* krävde att något skulle skickas eller avlyssnas, skulle driverna prioritera det i största utsträckning utan att bryta mot interna timings eller logik som äventyrade förbindelsen med dess sammanlänkade fysiska enhet. Driver modulerna utvecklades som en klass för att förenkla skalbarheten av systemet.

Framer var modulen som satte ihop paketen. Den hade information om hur paketet ska utformas och hur hanteringen av escapetecken sker. Den gjorde också all beräkning av paritetsbit för att filtrera bort paket som inte var kompletta. För att skapa paket med innehåll som matchade protokollen skapades en tillhörande payload modul. Payload returnerade meddelandets innehåll till framern som paketerade det och skickade det till com-modulen.

Figuren 3.3 visar hur drivern startar sekvensen när något skulle skickas. Vid inkommen data startades istället processen fast omvänt, där paketet från com delades upp och avläses av payload och framer. Informationen behandlades sedan av driver och sammanställdes i resultatsträngar. Resultatsträngarna från driversen returnerades tillsammans med systemtiden för testets start och sluttid upp till *main* för att loggas.



Figur 3.3: Flödesschema för hur en driver använder sina moduler för att kommunicera med en fysisk enhet.

Riktlinjer och praxis vid utveckling av mjukvara

Under programmeringen bestämdes det att följa *Clean code(Martin, 2009)* i största möjliga mån under programmeringen. Vilket betyder att varje modul ska utvecklas med endast ett syfte. Det var anledningen till att exempelvis UART-paketet var uppdelat i fyra moduler. Där varje del av de fyra modulerna hade sitt eget uppdrag som tillsammans utgör det kompletta paketet. Varje modul i paketet ska också namnges tydligt utifrån sitt syfte.

Vidare nämner även *Robert C. Martin* att varje metod ska ha ett syfte för att vara tydlig. Det var extra viktigt i stora moduler som driversen och *testcase*. Likt modulerna var det också väldigt viktigt att metoderna får tydliga namn som beskriver dess syfte. Därför skulle metoder namnges med längre namn som liknade en hel mening, för att verkligen förtydliga vad den hade för syfte samt att parametrarna fick tydliga namn som beskrev vad de innebär.

Slutligen för att ha ett program med pålitliga moduler var det viktigt att hantera *none-type*. Enligt *Robert C. Martin* var det därför extra viktigt att alltid behandla när saker vara tomma eller saknades, för att inte skicka problemet vidare till andra moduler. Ifall metoderna returnerat *none* vidare skulle det krävts att varje metod hade ett sätt att hantera det, istället för att metoden närmast problemet löser det och omvandlar till ett mer användbart format.

Vid namngivningen har examensarbetarna strävat efter att följa Pythons språkriktlinjer [19]. Alltså har indrag varit fyra mellanrum, namn har i första hand varit små bokstäver och konstanter har varit stora bokstäver med understreck. Huvudsakligen skedde formateringen automatiskt i IDE:n där koden skrivits och i efterhand blivit kontrollerad av examensarbetarna.

3.1.3 Utveckling av UART-paketet

Efter att kodstrukturen var bestämd var UART-paketet det första som implementerades. Det utvecklades enligt figur 3.3. Eftersom det fanns behov för flera fysiska enheter skedde kommunikationen med en *uartdriver* per fysiskt enhet. Den modul som jobbade närmast enheter var *uartcom* vilket var bunden till en seriell adapter via USB. Inställningarna till *uartcom* kom från företagets protokoll. Där fanns bitar per sekund, bitfönster, stopbitar och fördröjning specificerat. Med hjälp av det färdiga Python-biblioteket *serial* skickades och lästes information med metoderna *.write()* och *.read()*.

Problemet som uppstod var att informationen som skulle skickas var i ett annat format än *uartcom* behövde för att skicka. *Uartcom* skickade rena bitar formaterat till en sträng samtidigt som informationen från *uartdriver* kom i hexadecimalt formaterat till integer. För att översätta hexadecimalt till binärt skapades *uartframer*.

I *uartframer* utvecklades en metod för att konvertera hexadecimala värden till strängar och sammanfoga dem i rätt ordning enligt protokollet. Sekvenserna sparades sedan binärt i strängar som paket redo att skickas över *uartcom*. Eftersom *uartframer* endast var ett tomt paketskelett krävdes *uartpayload* för att fylla paketet med information.

I *uartpayload* skapades färdiga metoder för att utifrån indata returnera motsvarande utdata för händelsen. På grund av att *uartpayload* returnerade innehåll utan struktur kombinerades den med *uartframer*. Parametrarna till *uartframer.make_frame()* var en kombination av flera metodanrop från payload för att skapa allt innehåll till paketet.

Uartdriver var designad att initiera *uartcom* till att skicka information vid tillfälle med *serial.write()*, annars ligga och läsa kontinuerligt med *serial.read()*. När drivern hade läst av inkommande data skickade den in bitsträngen till *uartframer* som delade upp den den i delar. Sedan med hjälp av att köra *uartpayload* omvänt omvandlades delarna tillbaka till hexadecimala representationer. De hexadecimala talen för varje del sparades undan i *uartframer* som olika variabler. Anledningen var för att undvika att dela upp och översätta paketet varje gång ett metodanrop frågade om ny information i paketet.

3.1.4 Utveckling av JSON-paketet

Det här avsnittet beskriver utvecklandet av den delen av koden som kommunicerar över ethernet. Modulernas huvuduppgift var att behandla JSON och följa designmallen i figur 3.3. Det enda som avvek från mallen var att framer och driver blev ihopslagna till en gemensam modul. Anledningen var att mallen skapades under utvecklingen av UART och var därför inte helt anpassad för JSON-paketet. När JSON användes så krävdes det betydligt mindre kod i modulerna samt att innehållet i *jsonframer* inte går att dela upp i två olika moduler på ett effektivt sätt.

Huvudmodulen i JSON-paketet var *jsondriver* vars uppdrag var att länka samman de andra modulerna. Drivern fick uppdrag från *testcase* och hade kod för att hantera de olika scenarion. Det innebar att först kalla på *jsonframer* för att generera information

att skicka. Det färdigformaterade meddelandet som returnerades togs emot av driver och skickades med *jsoncom*.

Modulen *jsonframer* innehöll all information för att formatera och avkoda JSON. Den innehöll även all information som krävdes för att kommunicera enligt företagets protokoll med basstationen. Modulen använde sig av *json.loads()* och *json.dumps()* när den gjorde omvandlingarna mellan Python och JSON. Framern hanterade de 8 datatyper som existerar i JSON samt escape-sekvensen vid specialtecken.

Under utvecklingen av *jsonframer* testades koden mot kända JSON-strängar för att se om den avkodade strängen matchade det den förväntades bli. Vid tillfällen där den inte matchade upprepades samma test fast med färre datatyper tills felet lokaliserats. Det upprepades flertalet gånger fram tills att *jsonframer* korrekt avläst inkommande meddelanden, samt korrekt formatterat JSON så att mottagaren accepterat meddelanden.

I *jsoncom* fanns endast en metod *.get_response()* som kallades ifrån *jsondriver*. Det var den metod som faktiskt kommunicerar över ethernet och skickar informationen formaterad i JSON. Den tog in det färdiga meddelandet som parameter och skickade det efter att ha skapat en session med basstationen. Därefter returnerades basstationens svar till *jsondriver*. Kommunikationen till basstationen med JSON utfördes med hjälp av biblioteket *json*.

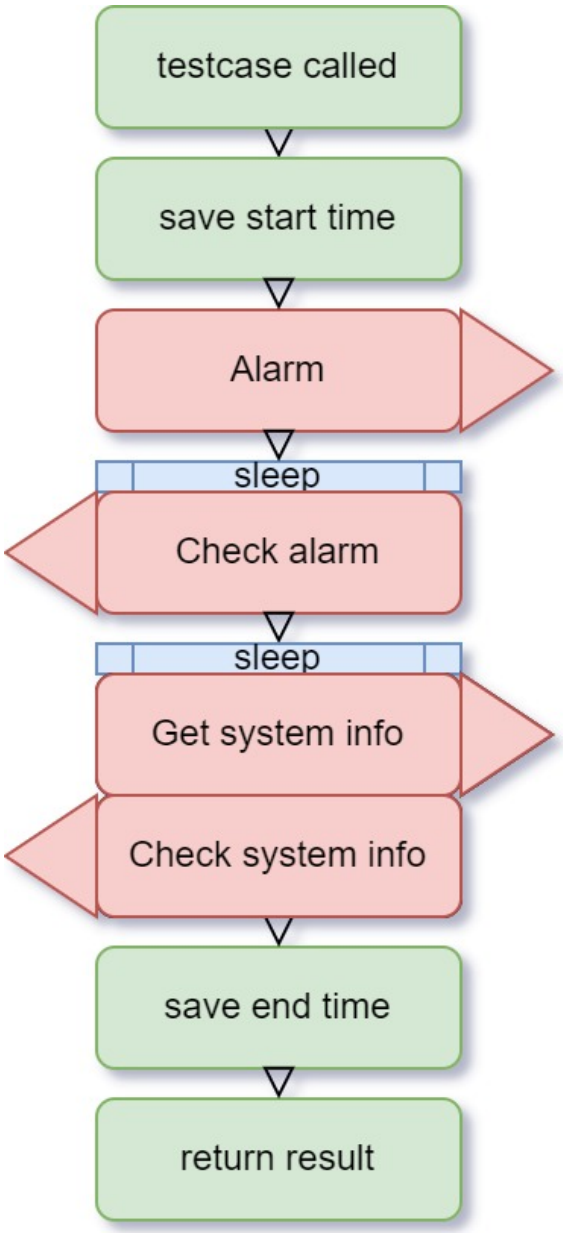
3.1.5 Utvecklingen av testfall

Testfall var ingenting som examensarbetarna behandlat tidigare, och med denna bakgrunden valdes det att först utveckla ett väldigt enkelt testförlopp som enbart testade en funktion. När testfallet visade sig vara pålitligt och fungera som tänkt, användes mycket tid på att göra testfallen flexibla. Examensarbetarna resonerade att istället för att alltid köra flertalet fasta testfall så tar *testcase* in parametrar, detta för att framtidsäkra koden och kunna ändra innehållet i testerna utan stora om-strukturingar.

Testfallet som skapades för att testa systemet illustreras i figur 3.4. Testets uppgift var i ett system där alla enheter var påslagna, generera och skicka ett larm till PIR-en. Därefter tilldelas enheten en rimlig tid att hantera larmet för att sedan läsa av drivern ifall ett larm inkommit. Nästa steg var att efter en längre tid läsa av systemets tillstånd och information och se ifall allting hade skett som det skulle. Alltså att larmet från enheten propagerat i nätet och hanterats av de andra enheterna.

Testfallet fungerade endast i ungefär hälften av de 100 testfallen. Samtidigt, om larmet saknades när drivern skulle läsa av det, fastnade applikationen i en oändlig loop och orsakade att tråden kraschade. Det gjorde att alla efterföljande testfall gav 0 resultat och att resultatsträngarna innehöll *NaN* i alla fält.

Det noterades att problemen uppstod på grund av flera separata problem. Det första problemet med att driver fastnar och tråden kraschar var kopplat till JSON-biblioteket som användes i *jsoncom*. Vid inläsning krävs det specifikation om antalet bitar som ska behandlas. Ifall bitarna var för få resulterar det i att bara den första delen av paketet



Figur 3.4: Testfallsstrukturen i första fasen.

som motsvarar de bitarna läses in. Därför hade siffran satts till ett väldigt högt tal i början av examensarbetet som garanterade att allt kom med. Men när larmet saknades, gjorde det att *jsoncom* försökte läsa in mer information än som existerade. Detta ledde till att den fortsatte att läsa och uppdatera indata utan att komma vidare då ingen mer data mottogs.

Lösningen till första problemet var att i *jsoncom* först läsa ett litet antal bitar som garanterat existerar i indatan. Denna siffra valdes till 10 eftersom det var mindre än alla storlekar på möjliga JSON-paket. Sedan skapades en kodsnutt i ett try-except-block för att behandla de inlästa bitarna. Ifall den lyckades hade hela paketet kommit fram. Ifall det fångades ett problem i except-klausulen innebar det att det fanns mer data att läsa in. Då försökte metoden istället läsa ytterligare 10 bitar och lägga till det till den totala strängen. Därefter försökte metoden igen behandla och läsa kontinuerligt

tills det passerat try-except-blocket utan att ett fel inträffat. På så sätt garanterade modulen att med små inläsningar alltid få ett komplett paket.

Det andra problemet var kopplat till att *uartdriver*n inte lyckats larma enheten korrekt. Ett exempel är när kommunikationen inte följer företagets protokoll. I ett försök att hitta orsaken lyftes hela uarttrådens paket ut i en separat mapp. Det gjorde det möjligt att testa larmsekvensen utan att ha flera felfaktorer inblandade. Testen där larmet isolerats gav inte ett tydligt felmeddelande, men det kunde konstateras att felet orsakades av applikationens kod och inte enheten. Larm som ska ha skapats av applikationen existerade inte på logikanalysatorn och var därför inte korrekt. Det innebar att kodfelen skapade falska negativa i testloggen.

Lösningen på andra problemet gick inte att hitta vid den här punkten in i examensarbetet. Vissa mindre ändringar gjordes som innebar att det hände mer sällan men det var fortfarande inte löst. De mindre ändringarna som gjordes innefattade att justera tiderna i *.sleep()*. För att hitta rätt tider studerades flera fungerande enheter med logikanalysator och interna loggfiler. Därefter valdes nya timeout och väntetider till *uartdrivern* och *uartcom*. De nya tiderna gjorde att felet påträffades ungefär 30 gånger under 100 testfall. Andelen falskt negativa utfall var då nere på ca 30% vilket gjorde att arbetet fortsatte med detta kvarstående fel. Felet dokumenterades för att lösas i framtiden och istället prioritera utvecklingsfas två.

3.2 Utvecklingsfas två

Efter första fasen var avklarad hade examensarbetet ett system som bestod av tre enheter. En Raspberry Pi, en basstation och en PIR. Applikationen kunde vid detta tillfälle generera larm från Raspberry Pi till PIR, som sedan larmade basstationen. Koden från första fasen användes sedan i fas två. Planen var då att utöka applikationen för att hantera ytterligare en enhet på UART samt lägga till stöd för att läsa av kamerans händelser via JSON och spela upp video för den via Raspberry Pi.

3.2.1 Videouppspelning för Scout Mk3

För att möjliggöra kommunikation med ytterligare en produkt, med minimal ändring i koden, beslutades det att behålla så mycket som möjligt av UART-paketet. *Uartdriver*n skrevs om som en klass för att ta in parametrar för vilken enhet den blir bunden till. Parametrarna skickades vidare i kedjan till payload och framer för att paketera informationen på korrekt sätt. Därefter skickades parametrarna vidare till *uartcom* för att konfigurera kommunikationsparametrarna.

Först undersöktes det ifall det gick att koppla två UART-enheter till en seriell adapter. På internet fanns olika information angående ifall det fungerar. Det fanns till exempel RS-485 som kunde hantera flera enheter [20]. Men eftersom det fanns fler USB-utgångar på Raspberry Pi samt att det fanns fler serielladaptrar på arbetsplatsen valde examensarbetarna att inte undersöka det vidare. En annan anledning var för att minimera risken för fel eftersom det finns mindre att testa på grund av återanvändning

av metoder i moduler. Eftersom alla UART-moduler var omskrivna som en klass, ändrades konstruktorn för att specificera om *uartcom* ska kommunicera med en enhet av typen PIR eller Scout.

Modulen som hanterar video var *videoplayer*. Den läser in sparade videos som ligger på Raspberry Pi lokalt. För att spela upp video användes biblioteket *OpenCV*. Det testades att spela upp video med *VLC media player* och *OpenCV*. Det fanns problem att få *VLC media player* att spela upp video på Raspberry Pi när användaren fjärranslutit via SSH. Därför testades *OpenCV* och där gick det att spela upp på dess skärm även om vissa inställningar gjorde att det visuellt såg fel ut. Därefter valdes *OpenCV* som biblioteket då det gick att spela upp och för att det hade funktioner där varje bild i videon gick att behandla var för sig.

Standardsättet att använda *OpenCV* var att visa nästa bild direkt när den var behandlad av koden. Det gjorde att bildhastigheten varierade och inte alltid stämde överens med originalet. Men det fanns ett attribut *cv2.CAP_PROP_FPS* för att läsa av videons orginalhastighet. Därefter kunde modulen vid varje ny bild lägga till en *time.sleep()* för att hålla en jämn och korrekt takt. Utöver hastigheten behandlades bilder på fler sätt. Det första var att den skalades om vertikalt och horisontellt för att matcha Scout-ens storlek. Det andra var att färgrymden ändrades för att följa vad Scout-en vanligvis använder.

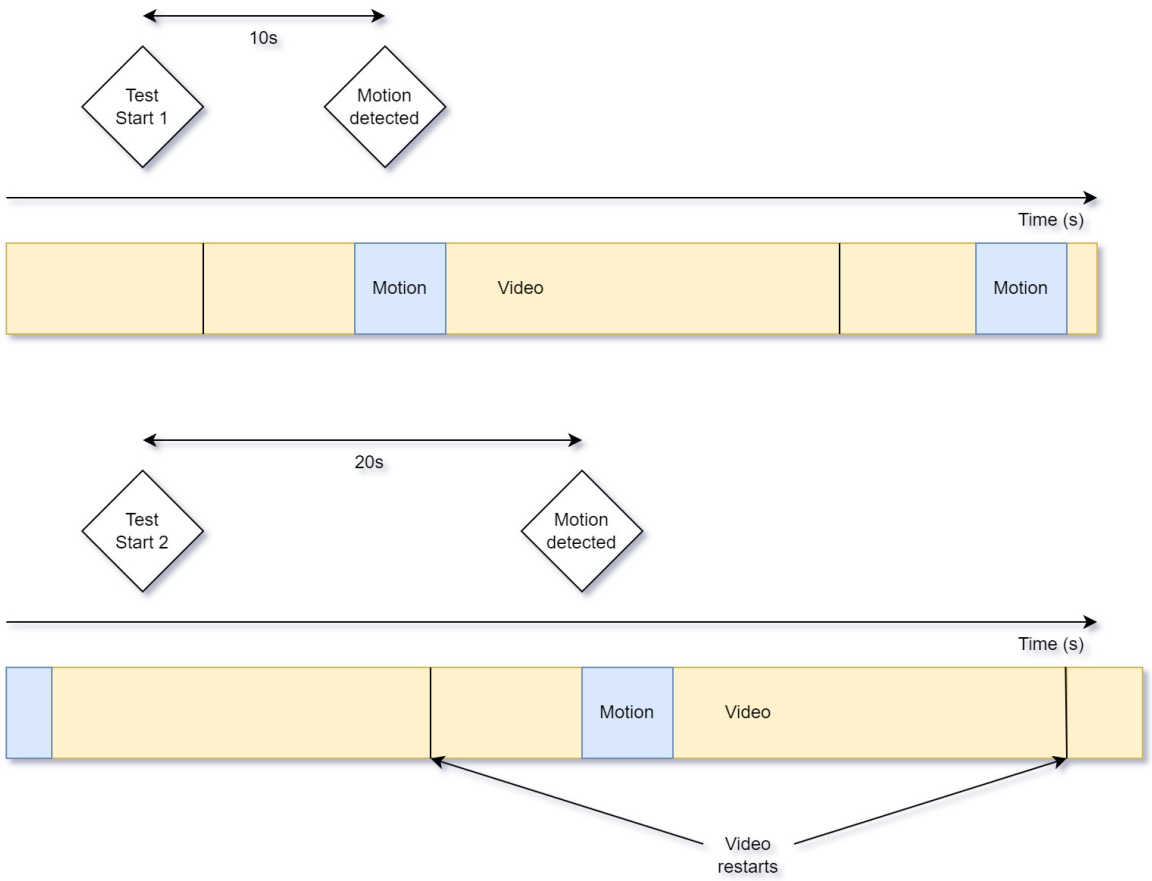
När programmet för att generera en videoström var klart skulle det undersökas hur hårdvaran skulle kopplas samman. Lösningen som examensarbetet tog fram var att koppla in kameran som en skärm till Raspberry Pi. Fördelen med det var att inget extra arbeta krävdes för att koppla in sig på Scout-en. Nackdelen var att eftersom hela Raspberry Pi gick via den utgången innebar det att skrivbordet och muspekare visades även när inget spelades upp.

Lösningen för att garantera att Scout-en ser rätt video var därför upprepad uppspelning av en video med *videoplayer* så länge Raspberry Pi var igång. Även om modulen körde videon i fullscreen gömdes inte muspekaren. Det utgjorde ett problem eftersom det påverkade testen då muspekaren kunde vara på olika platser i bilden vid olika tillfällen. Eftersom det inte fanns ett sätt att gömma muspekaren med hjälp av koden var lösningen istället att dra den ut ur bild.

Vid den här tiden fungerade videouppspelningen tillräckligt för att exekvera koden men den avvek från originalet. Anledningen var att tiden mellan varje bild i videon inte var konstant som videons original antal bilder per sekund. I koden fanns det en *.sleep()* mellan varje bild på 40 millisekunder för att uppnå 25 bitar i sekunden. Men det verkade som om att Raspberry Pi ibland nådde sin maximala belastning, vilket resulterade i att bilderna tog längre tid att hämta och därför spelades upp i en ojämn takt.

En lösning till detta var att kontrollera systemtiden vid varje inläsning av nästa bild och sedan vänta tills 40 millisekunder hade passerat innan bilden lades ut. Alternativ två som uppfattades som enklare var att separera videouppspelningen till en ny extra Raspberry Pi. På så sätt skulle den kunna spela upp video utan att störas av annan processanväning av testfallen på den huvudsakliga Raspberry Pi. Vid testfall

uppmättes att videouppspelningen använde ca 87% av CPU-kraften vilket talade för att använda en separat Raspberry Pi.



Figur 3.5: Visar hur en försening av videon kan ge missvisande tid för testfall.

Efter att videouppspelningen fungerade som förväntat uppmärksammades en ny förbättringsmöjlighet. Förbättringen grundades i att applikationen senare skulle mäta tiden för testfall, vilket gav kravet att testen måste vara så lika varandra och utföras så identiskt som möjligt. Vid det tillfället spelades videon om hela tiden och det var omöjligt att säga hur långt av videon som var uppspelad när ett nytt testfall skulle startas. Exempelvis likt figur 3.5 om det en gång startades 15 sekunder innan rörelse och 25 sekunder innan rörelse i andra fallet, skulle det ge en 10 sekunders skillnad. Den här skillnaden skulle få testfall två att till synes ha sämre responstid, fast den endast fått sämre förutsättningar men reagerat lika snabbt som testfall ett.

För att ta bort förseningen utvecklades en metod för att starta videon i samband med testfallet. Metodens funktion var att dra ett GPIO-stift högt när testfallet börjar. Sedan med hjälp av två andra GPIO-stift var det möjligt att välja vilken video som skulle spelas upp. Eftersom två stift kan ha fyra olika tillstånd var det möjligt att ha upp till fyra olika videos.

Det var den huvudsakliga Raspberry Pi som styrde stiften och videospelande Raspberry Pi som avläste. Videon startades vid stigande flank och om det redan spelades en video i modulen stängdes den av och den startades om igen. För att säkerhetställa att Scout-en hann se rörelse inom rätt tid, försköts videon ett par bilder för att rörelsen

alltid skulle ske samtidigt oavsett video. Antalet bilder som försköts per video sparades i konfigurationsfilen.

3.2.2 Utvecklingen av jsonthread

När kamerahuvudet var implementerat skulle det hämtas ut data från Scout-en. Efter att ha granskat tillgängligt material visade det sig att det inte fanns något enkelt sätt att extrahera datan. Något som noterades var att basstationen själv skickade ut uppdateringar vid nya händelser. Ett förslag till att spara ner all data var att kontinuerligt lyssna och filtrera basstationens uppdateringar och spara undan dem. Därefter var det möjligt för andra moduler i applikationen att hämta ut datan vid behov.

Utvecklingen av *jsonthread* implementerade förslaget i största möjliga mån. Den designades för att konstant läsa av basstationen och notera när bilder från Scout-en var klara. Det fanns en lista för varje JSON-datafält som kunde läsas av. Det var känt hur många bilder Scout-en skulle ta vid ett test efter som det bestämdes i konfigurationsfilen. Därför lades det till en spärr i *jsonthread* att aldrig acceptera fler datapunkter än bilder i sin listor. Det sågs också till att från *testcase* kalla på *.reset_files()* efter *testcase* sparat undan data för att tömma *jsonthread*s listor inför nästa testfall.

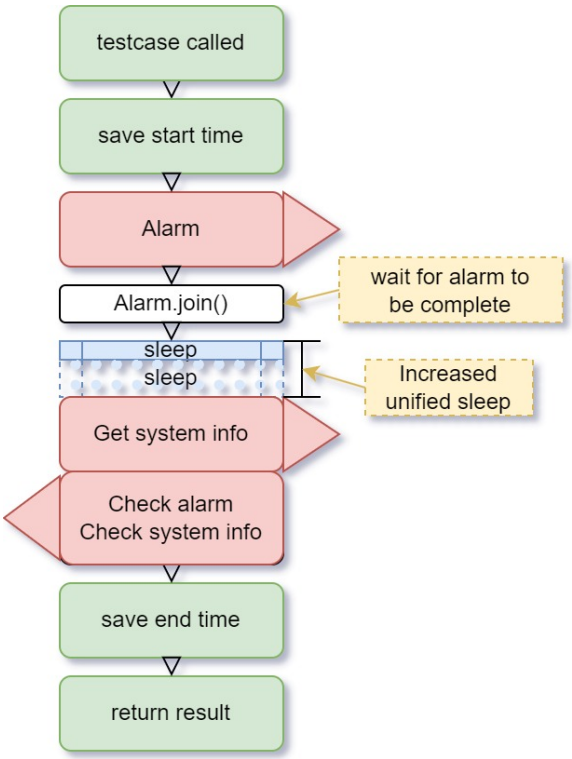
Vid test av *jsonthread* noterades det att den ibland kraschade när den läste av felaktig data. Eftersom orsaken inte hittades till att datan lästes in på fel sätt, bestämdes det att istället starta om tråden. Det lades till ett try-except-block runt alla funktioner och ifall något gick fel sattes en variabel till *true* från except-klausulen. Det gjorde att om *testcase* såg att variabeln var *true* inför nästa iteration stängde den ner tråden och startade en ny igen.

3.2.3 Vidareutveckling av testfall

Tidigare testfall drabbades av problem med falska negativa och falska positiva resultat. Mycket av detta berodde på att applikationerna var programmerad att följa vissa tidsintervall och fördröjningar som valts utifrån de enheter som testades. När produkterna sedan flyttades eller utsattes för störningar eller andra problem, introducerades variationer i tiden det tog att reagera på förfrågningar. Det ledde till att modulen ibland fick svar för snabbt och missade det, eller att det läste av information innan allt hade hunnit genereras.

För att lösa det här problemet infördes spärrar i de flesta modulerna. Spärrarna säkerställde att modulerna väntade i en while-loop tills de hade fått svar på sina frågor. Detta var nödvändigt eftersom applikationen använde flera trådar och vissa av dem fortsatte utan att ha fått någon data att behandla. Spärrarna såg därmed till att alla trådarna var synkade för att inte läsa in tomma variabler. Ett exempel var *Alarm.join()* i figur 3.6.

Implementeringen av spärrarna gjorde det mer pålitligt men det tog också längre tid. Anledningen var att när *testcase* väntade tills den garanterat fått med allt resulterade



Figur 3.6: Testfallsstrukturen i andra fasen

det i att vissa metoder väntade upp till fyra gånger längre. För att lösa det problemet infördes en kombinerad sleep synligt i figur 3.6, som började när spärren garanterat ett larm. Därefter var det säkert att hämta all systeminfo på en och samma gång då garanti fanns för att systemet var färdigpropagerat. Fördelen var att istället för att vänta på alarm och systeminfo separat gjordes det nu tillsammans vilket sparade tid. Det ökade också pålitligheten av resultatet eftersom nu kontrollerades resultatet endast när modulen kunde verifiera att ett larm skett, samt att modulen tittade på datan efter en paus vilket gjorde att den inte missade eftersläpande data.

3.3 Utvecklingsfas tre

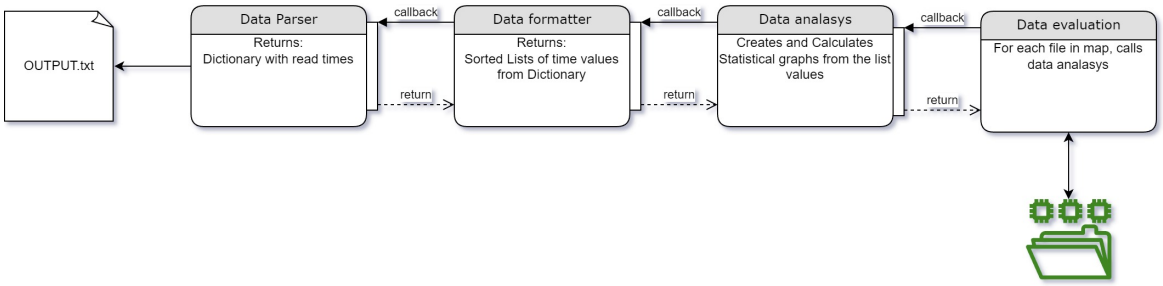
I den tredje fasen av examensarbetet påbörjades bearbetningen av den ackumulerade informationen som inhämtats under långtidstestet. Resultatet från detta test lade grunden till applikationens poängsystem då detta test blev det längsta och mest omfattande testet som genomfördes.

Som tidigare nämnts returneras testresultaten till en textfil som sparas i en valfri mapp. Textfilen uppdateras kontinuerligt under testets gång och var känslig för manipulering. En oavsiktlig input från användaren kan orsaka att modulen förlorar skrivrättigheter till filen, vilket resulterar i förlust av värdefull data.

För att säkerställa att den statistiska uträkningen inte ska påverka pågående tester, designades modulen för att ha minimal påverkan på filen genom att ingen information lades till eller hämtades ut på ett sätt som påverkar skrivrättigheterna i filen. För att

behandla resultatfilen hämtas de relevanta värdena ut och för att sedan returneras till andra moduler där den formaterade datan och används som input till diverse statistiska modelleringar.

3.3.1 Utvärdering och statistik



Figur 3.7: Visualisering av programhiearkin för den statistiska analysen.

Den här delen beskriver utvecklandet av modulerna i figur 3.7 som hade hand om den statistiska modelleringen. För att köra utvärderingen så skapas en modul vid namn *dataevaluation*. Denna modul var utformad för att analysera samtliga loggfiler som finns i en specifik katalog genom att anropa de nedan beskrivna modulerna för varje *.txt*-fil i sekvensiell ordning.

dataparser

Denna modul var utformad för att jobba närmast loggfilen och hade som uppgift att analysera samt extrahera relevant data om de olika tidpunkterna som sparas ner i loggfilen. Modulen innehåller en uppsättning av funktioner och datastrukturer som används för att läsa, bearbeta och lagra den extraherade informationen.

Informationen lagras i en dictionary som lagrar nyckel-värdepar. Nyckeln för testfallen var testnumret, och värdena var olika tidpunkter som används för beräkningar samt listor som beskriver relationen mellan dessa tidpunkter. För varje test anropas flera metoder som läste av och returnerade de sökta värdena.

Reguljära uttryck, även kallade Regex, användes i denna modul för att identifiera och extrahera specifika delar av texten i loggfilen. Regex-funktionerna sökte efter mönster och matchade textsträngarna med dessa mönster. Med användning av Regex blir inverkan på loggfilen minimal, samtidigt som huvudmodulens läs- och skrivrättigheter garanteras. Det gav även en robust och skalbar lösning för att hantera och analysera loggfiler med varierande struktur och innehåll.

I denna modul definieras flera Regex-mönster för att matcha olika delar av loggfilen, som starttid för loggen, testdata, kamerainformation samt när systemets olika delar aktiverades. Dessa mönster användes sedan i en loop för att iterera igenom loggfilen rad för rad. Vid varje matching av ett mönster extraherades de önskade värdena från textsträngen och lagrade de i en dictionary för vidare bearbetning och analys. Modulen returnerade en dictionary samt det avlästa testets datum.

dataformatter

Denna modul hämtar informationen från föregående moduls returnerade dictionary och skapar tre sorterade listor. Detta för att i nästa steg kunna räkna ut det statistiska värdet median. De sorterade listorna returnerades tillsammans med datumet för testet och behandlades i nästa steg.

dataanalasys

Nästa modul i hierarkin var avsedd för att analysera och visualisera datan som returnerats som listor från föregående modul. För att utföra dessa uppgifter så inhämtas en rad bibliotek och moduler, inklusive *Matplotlib, Numpy, Statistics, Csv* och *Os* för att utföra beräkningar, läsa filer och skapa grafer. Följande beräkningar utförs med hjälp av *statistics*:

- Minsta och största värden i listorna
- Medelvärde, som representerar det centrala tendensvärdet för tidpunkterna
- Median, som indikerar mitten av tidpunkterna när de var sorterade i stigande ordning
- Kvartiler, som ger en indikation av spridningen och hur tidpunkterna var fördelade
- Standardavvikelse, som beskriver hur spridda tidpunkterna var från medelvärdet
- Normalfördelning, som är en statistisk modell för att beskriva hur tidpunkterna fördelas kring medelvärdet

Genom att utföra dessa beräkningar kan modulen ge en detaljerad och informativ översikt över testfallen, vilket gav användaren möjlighet till att på ett enklare sätt utvärdera och dra slutsatser om systemets prestanda. Utöver att genomföra beräkningarna så visualiserades även informationen i form av normalfördelningskurvor, felstapeldiagram och andra anpassade grafer som gav ytterligare insikt i datan. Slutligen lagrades resultaten från de föregående beräkningarna samt de visualiserade graferna i CSV-filer och PNG-filer.

3.4 Utvecklingsfas fyra

Utvecklingsfas fyra gick ut på att förbättra och förfina applikationen ytterligare. Fokus under fas fyra var att göra applikationen mer stabil och optimerad för användaren. Detta innebar att vidareutveckla JSON-paketet samt att implementera ett nytt sätt att logga informationen.

3.4.1 Omskrivning JSON-paketet

Under utvecklingsfas fyra visade sig att antaganden som tidigare gjorts inte stämde för *jsonthread*. Då uppmärksammades ett mer pålitligt sätt att hämta ut data från basstationen. Detta gick att göra på samma sätt som moduler använt för att verifiera PIR-larmet. Med den nya informationen var det tydligt att *jsonthread* var bristfällig och opålitlig då den inte alltid gav rätt svar. Det var vanligt att den hängde sig utan att startas om även om funktionen för omstarten var implementerad.

Examensarbetarna gick över från en metod där systemet kontinuerligt läste av informationen i basstationenen, till en mer effektiv och målinriktad lösning. Den implementerade lösningen ställde specifika frågor via API-et där basstationenens svar användes. På detta sätt kunde användaren konfigurera modulen genom att ändra hur lång tid efter ett larm som frågan ställs. Detta för att försäkra sig om att det fanns ett svar i basstationen innan information från den försökte hämtas ut.

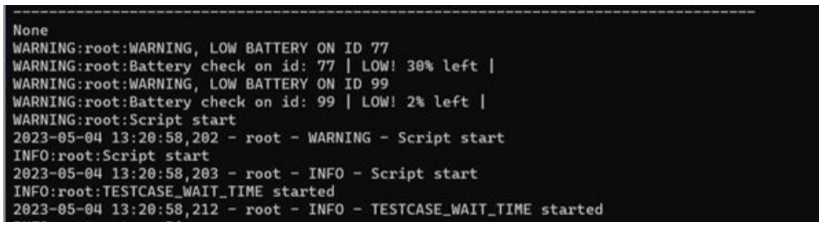
Vidareutveckling av JSON var att implementera nya JSON-strängar som hämtade ut ny information, som exempelvis batteriprocent och systemklocka. Modulen informerade användaren om hur batterinivån såg ut för de olika enheterna vid teststart, samt såg till att alla enheter fick tilldelat samma systemtid som Raspberry Pi. Anledningen till synkroniseringen av klockor var för att kunna analysera de olika tidsintervall som mättes i systemet, och anledningen till användandet av Raspberry Pi:s klocka var dess implementering av Network Time Protocol vilket var mycket användbart i examensarbetarnas tillämpning då användandet av olika klockor hade bidragit till oönskat jitter [21].

Slutligen implementerades JSON-strängen som uppdaterade vilken enhet som skall väckas vid händelse av larm. Denna larmkedja går att ändra i konfigurationen och bidrar till fler möjligheter att bygga upp system och utvärdera dem. Enheterna som användes i strängen kom från konfigurationsfilen för att göra det möjligt för användaren att variera dem.

3.4.2 Implementering av debugging

Tidigare i applikationen skrevs all information ut med hjälp av *.print()* vilket gjorde att det var enormt mycket meddelanden i konsolen. Det resulterade i att viktiga meddelanden missades av användaren och att oviktiga meddelanden som kommunikation tog all uppmärksamhet. För att lösa problemet ersattes *.print()* med metoder från *logging* biblioteket. Varningarna dokumenterades med hjälp av *.warning()* och vanlig information att programmet fungerade loggades med *.info()*. Utöver det behandlades exceptions med *.error()* och alla metoderna inkluderade tillsammans med strängen en tidstämpel ner på en noggrannhet av en hundradels sekund.

Fördelen med *logging* var möjligheten att filtera vilka meddelande som syns. Det gick till exempel att filtrera bort allt som inte var *error* vilket gjorde det lätt att hitta, felsöka och lösa problem. I figur 3.8 visas en ofiltrerad konsoll med både batterivarning och information.



Figur 3.8: En ofiltrerad konsollutskrift som visar batterivarningen.

Ifall gränsen ställs in på *WARNING* skulle användaren endast få information om batterivarningen men inte utskriften att applikationen startade sin väntetid.

3.5 Utvecklingsfas fem

Den femte fasen skiljer sig från tidigare faser. Den var inte en påbyggnad på kod från föregående faser, utan utvärderade det som hade utvecklats i de föregående faserna. Fas fem använde sig av det kompletta systemets tider och poängsatte det för att lättare kunna genomföra olika testsessioner.

3.5.1 Utvärdering med hjälp av poäng

När testsystemet fungerade och metoder för analysering av loggen var implementerade var det dags att lägga grunden för hur testerna skulle jämföras med varandra. Det examensarbetarna kom fram till var att använda sig av ett poängsystem. Genom att implementera detta får användaren ett värde som kan jämföras med föregående tester och därav en mer överskådlig bild i hur systemet fungerar med den aktuella bestyckningen. Examensarbetarna valde att använda sig av AHP-träd, detta på grund av dess förmåga att låta användaren själv bestämma av vilken vikt de olika delarna i testet hade för påverkan av det slutgiltiga resultatet.

Datan från testfallen gjordes om med hjälp av en modell som examensarbetet tog fram. Tabellen för hur värdena som användes i AHP-trädet beräknades visas i tabell 3.1. Det bestämdes att uträkningarna skulle resultera i en siffra mellan 0 och 1 för att göra det enkelt att följa. Därför var beräkningarna konstruerade på ett sådant sätt att varje datavärde var mindre än 1.

Företaget hade uttryckt sin önskan om att deras verkliga data inte ska offentliggöras. Därför hade examensarbetarna skapat en uppsättning fiktiva, men representativ data som kan liknas med systemet. Det är den datan som användes i resterande beräkningar i kapitlet.

3.5.2 Designen av AHP-trädet

Som tidigare nämnts implementerades ett AHP-träd genom att knyta samman figur 3.1 i formen av ett träd. Detta gjordes enligt figur 3.9 där all indata var kopplad till

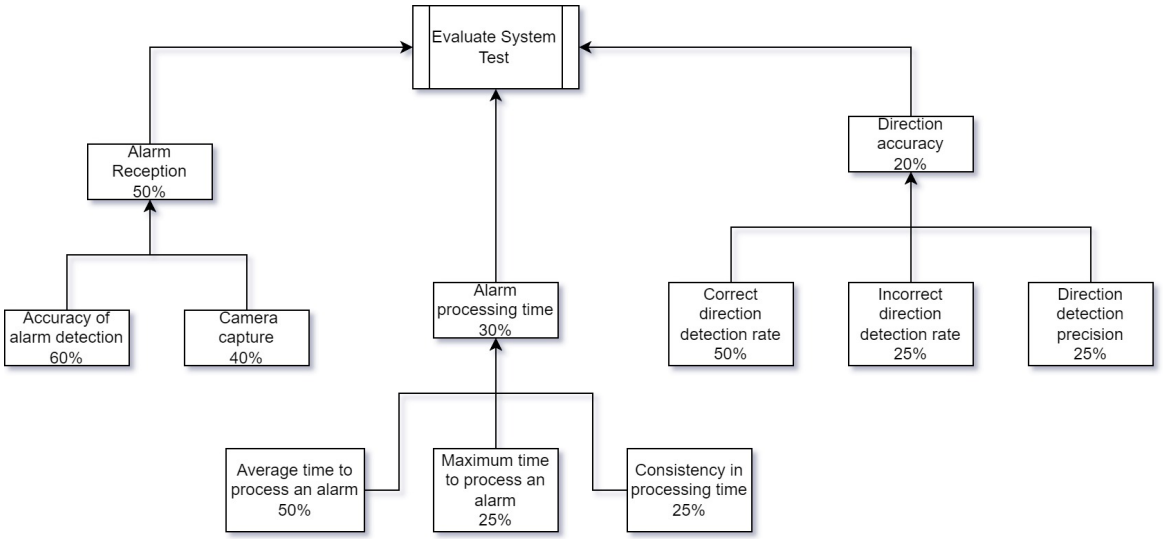
Typ	Förklaring
Antal lyckade larmdekteringar	Andelen lyckade larm i procent dividerat med 100
Antal lyckade bilder	Andelen lyckade bilder i procent dividerat med 100
Genomsnittlig tid för att behandla ett larm	Antal tidsenheter som ligger inom den bestämda avvikelsen från den normalfördelade tidens medelvärde dividerat med 100
Antal larm som tog över den maximala tiden	Antal larm som ligger utanför den översta kvantilen av larmtider dividerat med 100
Konsekvens i behandlingstid	Andelen av tidsenheter som faller inom den mellersta kvartilen, dividerat med 100.
Korrekt riktningsdetektering	Antalet larm med rätt riktning dividerat på antalet fel, dividerat med 100
Felaktig riktningsdetektering	Antalet larm med fel riktning dividerat på antalet rätt, dividerat med 100
Riktningsdetekteringsprecision	Andelen larm med rätt riktning i procent dividerat på 100

Tabell 3.1: Förklaring av varje typ av indata i AHP-trädet och hur det räknas fram i examensarbetet.

den slutgiltiga poängsättningen. Huvudkriterierna bestämdes för att det ska gå att avgöra ifall systemet var stabilt, snabbt och pålitligt.

Det första kriteriet avsedde stabiliteten och bedömmer andelen lyckade larm jämfört med testfall. Det andra kriteriet avsedde hastigheten och bedömde tiden det tog för ett larm att propagera i systemet. Det tredje kriteriet avsåg pålitligheten och bedömde hur sanningsenligt informationen i larmet var. Första kriteriet var det mest avgörande då det betygsätter ifall systemet faktiskt lyckats generera larm och får därför en vikt på 50%. Resterande kriterier var mindre viktiga och de avgjorde hur bra alarmen var, därför får det andra 30% och slutligen det sista 20%.

Lyckade alarm var den viktigaste datan som inkommer från testen. Eftersom det inte spelade någon roll hur lång tid eller vilken information larmet hade ifall det aldrig lämnade enheten. För att avgöra antalet lyckade larm hade examensarbetarna delat upp det här huvudkriteriet i två delkriterier. Det första delkriteriet *Lyckade larm* stod för andelen lyckade jämfört med missade larm från PIR-en. Det andra delkriteriet



Figur 3.9: AHP träd med deras vikt

var *Lyckade bilder* och det avsedda antalet lyckade bilder från Scout-en jämfört med missade. På grund av att PIR-ens larm i alla fall varnar användaren om att något händer och därför anses var något hjälpsam, fick det en större vikt. Dess larm får 60% medans Scout-ens tagna bilder får en vikt på 40%.

Tiden det tar för ett larm att komma fram var inte det viktigaste för användaren, men ifall det kommer alldeles för sent kommer användaren inte längre ha någon nytta av det. För att avgöra hur bra alarmet propagerar i nätverket hade examensarbetarna satt upp tre delkriterier. Det första delkriteriet får vikten 50% baseras på den genomsnittliga tiden det tar för larmet att komma fram i systemet. Resterande 50% fördelades jämnt mellan maxtiden för ett alarm på 25% och avvikelse från medel.

Riktningens tillförlitlighet delades upp i ytterligare tre delkriterier. Det första på 50% var ifall alarmet kom med rätt information gällande riktning. Ifall alarmet kom med felaktig information sattes vikten istället till 25%. Slutligen fick andelen korrekta riktningar vikten 25% eftersom det resultatet redan reflekterades i de andra två delkriterierna.

Matematiska uträkningen i AHP-trädet

Matematiken bakom Analytic Hierarchy Process innebar beräknadet av en summa poäng baserat på prestandan för varje testfall med hjälp av delkriterier. Vikten av kriterierna angavs i procent och representerade den relativa betydelsen av varje kriterium och delkriterium. Genom att beräkna hur ofta ens delkriterier uppfylls, kan sedan statistiken användas för att skapa en poängsättning.

Efter ett korrekt test gavs värden enligt tabell 3.2. Utifrån första testets värden multipliceras prestandarvärdet för varje delkriterum med dess motsvarande vikt för att beräkna den viktade poängen. Detta gjordes för alla delkriterier under varje huvudkriterum. Resultatet summerades ihop och multiplicerades sedan med den förbestämda

Typ	Test ett
Antal lyckade larmdekteringar	0.9
Antal lyckade bilder	0.1
Genomsnittlig tid för att behandla ett larm	0.8
Antal larm som tog över den maximala tiden	0.05
Konsekvens i behandlingstid	0.85
Korrekt riktningsdetektering	0.88
Felaktig riktningsdetektering	0.12
Riktningsdetekteringsprecision	0.9

Tabell 3.2: Systemets värden och utfall efter första testet.

vikten för varje huvudkriterium. Slutligen summerades de viktade kriteriepoängen för att få det totala poängen för testfallet i ekvationen nummer 3.4:

Kriterium 1:

$$0,9 \cdot 0,6 + 0,1 \cdot 0,4 = 0,58 \tag{3.1}$$

Kriterium 2:

$$0,8 \cdot 0,5 + 0,05 \cdot 0,25 + 0,85 \cdot 0,25 = 0,625 \tag{3.2}$$

Kriterium 3:

$$0,88 \cdot 0,5 + 0,12 \cdot 0,25 + 0,9 \cdot 0,25 = 0,695 \tag{3.3}$$

Totala poäng:

$$0,58 \cdot 0,50 + 0,625 \cdot 0,3 + 0,695 \cdot 0,2 = 0,6165 \tag{3.4}$$

3.5.3 Omstrukturering i AHP-trädet

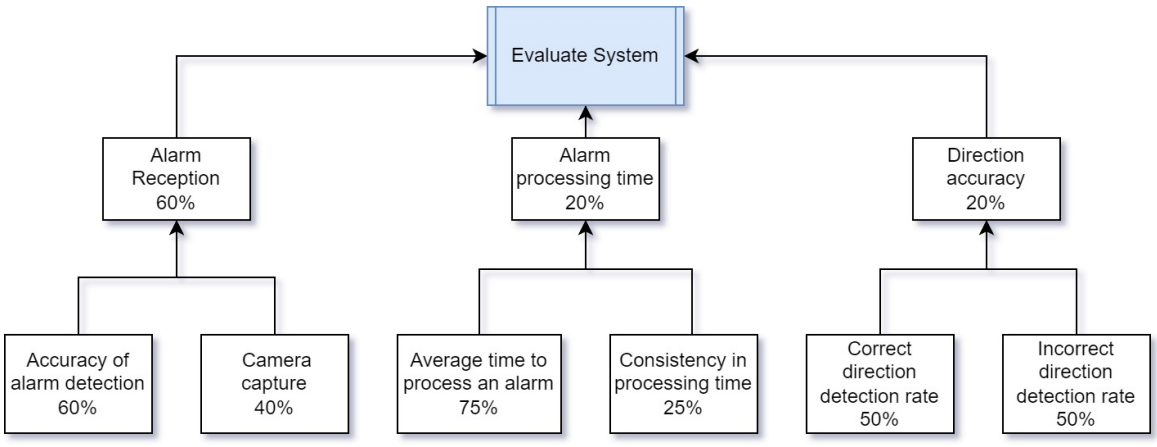
Examensarbetarna använde sig av trädet beskrivet ovan för att testa sina teorier. Under dessa tester uppkom tankar om hur en vidareutveckling av trädet hade sett ut. Detta tillsammans med dialog med företaget resulterade i omstrukturering av det befintliga trädet. Den första ändringen var den verkliga tyngden av ett alarms mottagning. Då systemet vart helt beroende på att ett larm registreras för att kunna gå vidare i flödesschemat, valde examensarbetarna att justera vikten för alarm mottagning med en ökning på 10 procentenheter. Vidare resonerar examensarbetarna att om tiden var för lång, alternativt om riktningen var fel, så bör detta värderas som lika problematiskt. Därför justerades även Alarm processerings tiden ned med 10 procentenheter.

Slutgiltligen togs den sammalagda informationen om detektions precisionen bort för att den redan reflekterades i korrekt och inkorrekt antal larm. Utöver det försvann den maximala tiden för att processera ett larm bort eftersom det bättre bedöms av medeltiden samt avvikelsen. De 25 procentenheterna som togs bort fördes över till medeltiden för hanteringen av ett larm. Eftersom att totala tiden för hela kedjan var

Typ	Förklaring
Antal lyckade larmdekteringar	Andelen lyckade larm i procent dividerat med 100
Antal lyckade bilder	Andelen lyckade bilder i procent dividerat med 100
Genomsnittlig tid för att behandla ett larm	Antal tidsenheter som ligger inom den bestämda avvikelsen från den normalfördelade tidens medelvärde dividerat med 100
Konsekvens i behandlingstid	Andelen av tidsenheter som faller inom den mellersta kvartilen, dividerat med 100.
Korrekt riktningsdetektering	Andelen larm med korrekt riktning dividerat med 100
Felaktig riktningsdetektering	Andelen larm med felaktig riktning dividerat med 100

Tabell 3.3: Förklaring av varje typ av indata i det slutgiltiga AHP-trädet och hur det räknas fram i examensarbetet.

av större intresse än avvikelsen i sekunder. Resultatet av dessa ändringar resulterar i trädet i figur 3.10.



Figur 3.10: AHP-trädet efter omskrivning av kriterier och vikter.

3.6 Källkritik

Följande avsnitt diskuterar examensarbetets källor och utvärderar deras trovärdighet.

3.6.1 Dokumentation

Examensarbetarna använder källor i form av programspråkens officiella dokumentation. Genom att använda sådana källor säkerställs att vedertagna branschstandard och bästa praxis används då sidorna kontinuerligt uppdateras.

Vid användning av moduler gäller det att vara uppmärksam då upphovspersonen inte nödvändigtvis behöver ha utvecklat modulen på korrekt sätt. Med bibliotek som *Numpy* och *Matplotlib* bedömdes det säkert att använda då det existerar en stor användarbas bakom som hade använt och analyserat modulerna under lång tid. Det var värt att notera att i samma takt som modulerna blir mer specifika, krymper även användarbasen och modulen blir mindre beprövad. I examensarbetarnas fall så används grundläggande metoder för att hantera data, och hade på det sättet försäkrat sig om att modulerna var beprövade.

Följande källor tillhörde kategorin dokumentation:

[1][2][4][3][7][5][6][9][16]

3.6.2 Hemsidor

Hemsidor ska man hantera varsamt då informationen kan skrivas av vem som helst och då det inte nödvändigtvis finns en redaktion bakom som kontrollerar innehållet. Därför har informationen från dessa källor korsrefererats med andra källor för att säkerställa innehållet. Examensarbetarna har använt dessa källor i första hand för att beskriva grundläggande koncept i hur tekniken fungerar samt dess uppbyggnad.

Följande källor tillhörde kategorin hemsidor:

[8][22][20][21][17][18]

3.6.3 Företaget

Källorna från företaget bedöms av examensarbetarna som helt pålitliga då dokumentationen och information som hämtats under arbetets gång har hämtats från utvecklingenheten som skapade produkterna. Företaget var det examensarbetarna har arbetat närmst med och har därför förlitat sig helt på deras dokumentation då ingen annan sådan existerar. Då examensarbetet drivs i företagets namn finner examensarbetarna att företaget var mån om att dokumentationen var sanningsenlig och korrekt.

Följande källor tillhörde kategorin företaget:

[11][12][13][14]

3.6.4 Böcker

Böcker som *Clean code(Martin, 2009)* och *The analytic hierarchy process—what it is and how it is used(Saaty, 1987)* anses pålitliga källor tack vare författarnas expertis, noggrann granskning, omfattande redigeringsprocess före publicering samt erkännande från forskarsamhällen. Dessa böcker refereras i flertalet olika akademiska skrifter och har på så sätt bevisats att var tillförlitliga och väl ansedda källor.

Följande källor tillhörde kategorin böcker:

[23][10]

KAPITEL 4

Resultat

I detta kapitel presenteras resultatet av applikationen som har utvecklats under projektets gång. Det beskriver även hur det slutliga tillståndet av applikationen såg ut och hur det presterade. Till slut presenteras den slutgiltiga poängsättningen som använts.

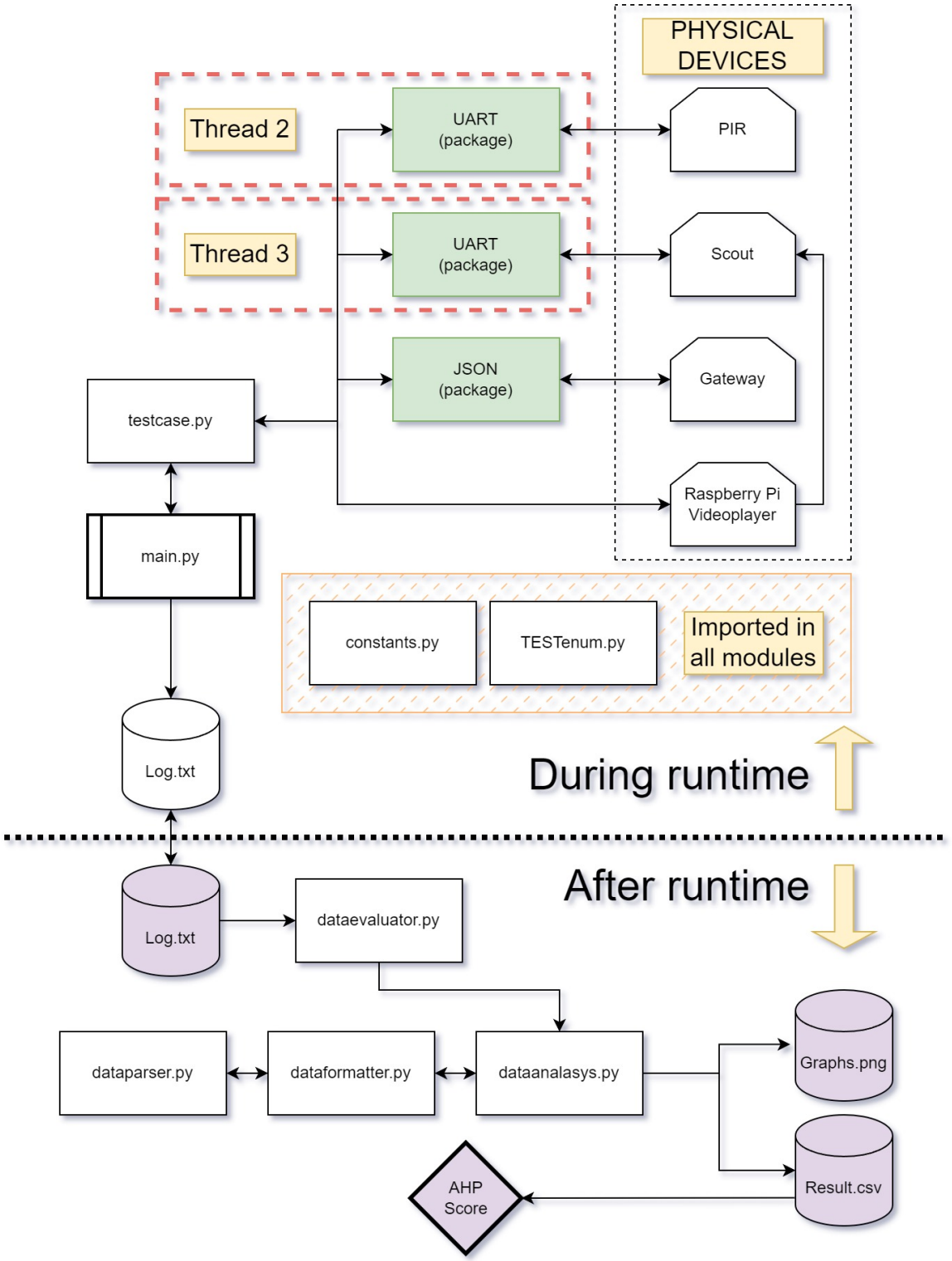
4.1 Applikationen

Den slutgiltiga applikationen visas i figur 4.1. Utöver *main*-modulen krävdes två trådar för ett fungerande program. Den övre delen av figuren utvecklades under fas ett, två och fyra. Statistik och resultat som visas i den nedre delen utvecklades under fas tre och fem eftersom det behövdes data att räkna på.

En sammanfattning av hur pålitligt den utvecklade applikationen är visas i tabell 4.1. Det beskriver även hur procenten felaktigheter har minskat över tid. Minskningen var ett mått på hur bra omskrivningar och optimeringar har fungerat i projektet eftersom det visade hur ofta koden var accepterad. Det slutgiltiga testet som kördes den 21 april 2023 gav att applikationen endast får någon typ av fel vid 4% av testfallen. Denna siffra uppmättes som högst vara 32% tidigare i mars. Det var inte implementerat en tillräckligt detaljerad log för att sätta någon procent på testfall tidigare än 21 mars 2023.

De slutgiltiga UART- och JSON-paketen utvecklades att följa mallen i figur 3.3 vilket resulterade i figur 4.2 och 4.3. Resultatet av utvecklingen bevisar att UART- och JSON-paketen kan kommunicera med systemets alla delar. De följer båda två företagets protokoll och specifikationer vilket gör att applikationen har möjlighet att testa alla enheter. Förbättringen i dessa paket är det som sänkt andelen fel i figur 4.1. Eftersom att majoriteten av problemen uppstått när meddelande skickats eller tagits emot på felaktigt sätt, samt om meddelandet förvrängts under transport. Slutgiltligen har UART- och JSON-paketen implementerat metoder som på ett stabilt sätt hanterat och återgått till ursprungsförloppet när dessa fel stötts på.

Sambandet mellan de två Raspberry Pi visas i figur 4.4. Utvecklingen resulterade i

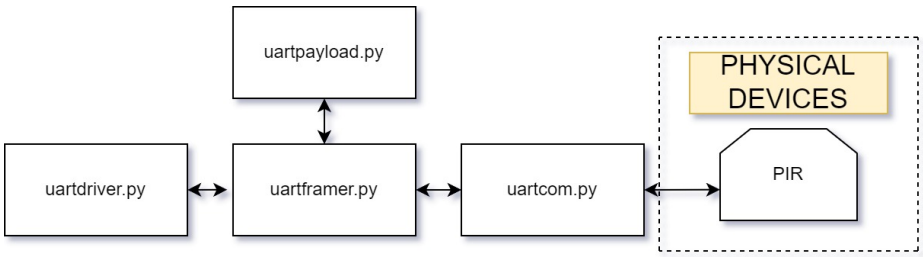


Figur 4.1: En fullständig överblick av applikationen i sin helhet. Den övre delen beskriver vad som skedde kontinuerligt under tiden testfallen kördes. Utöver det visar nedre delen vad som skedde när testet var klart och användaren ville se resultatet.

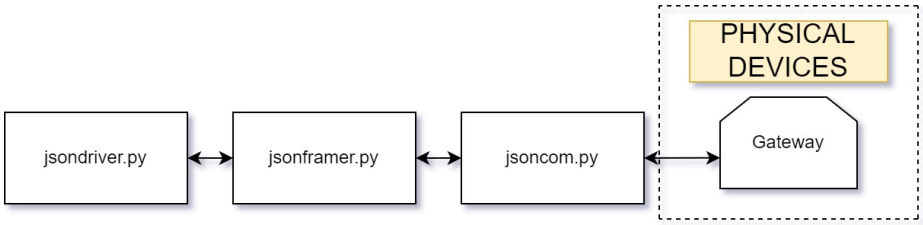
att Raspberry Pi kommunicerade binärt med hjälp av GPIO-stiften. Utifrån vilka stift som var höga eller låga valdes en av de videosen som var sparade lokalt för att spelas upp till Scout-en. Avläsningen av stiften skedde vid stigande flank på ett speciellt stift

Testfall med hela systemet				
Typ	21-25 mars	25-26 mars	19-20 april	21-23 april
Antal testfall	3242	714	474	2114
Total tid	112,5h	25h	16h	70h
Antal testfall som misslyckats på grund av projektets kod	1038	207	15	85
Resultat	32%	29%	3,2%	4%

Tabell 4.1: Testresultaten efter långtidstesters utveckling över projektets gång.



Figur 4.2: Beskriver hur modulerna i UART-paketet hänger ihop och kommunicerar med PIR-en.

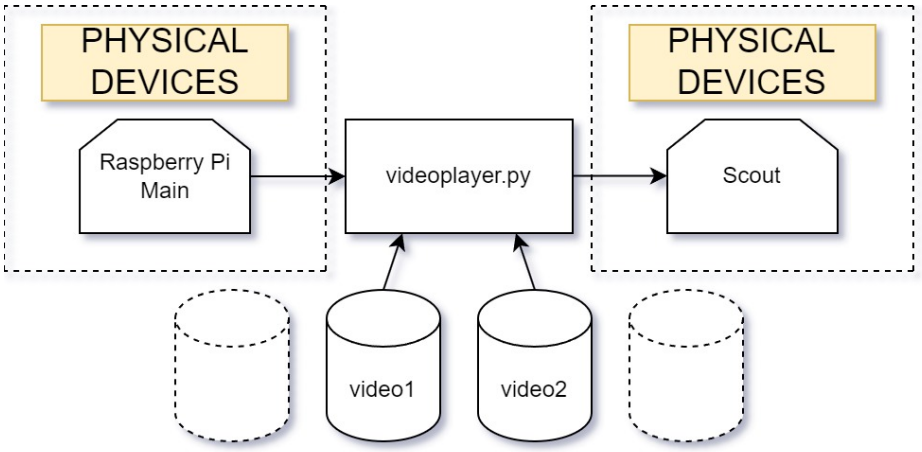


Figur 4.3: Beskriver hur modulerna i JSON-paketet hänger ihop och kommunicerar med basstationen.

som var synkroniserat med när det var dags att spela upp en ny video. Det här flyttade all form av bildbehandling och uppspelning till en ny Raspberry Pi vilket avlastade den som körde testapplikationen.

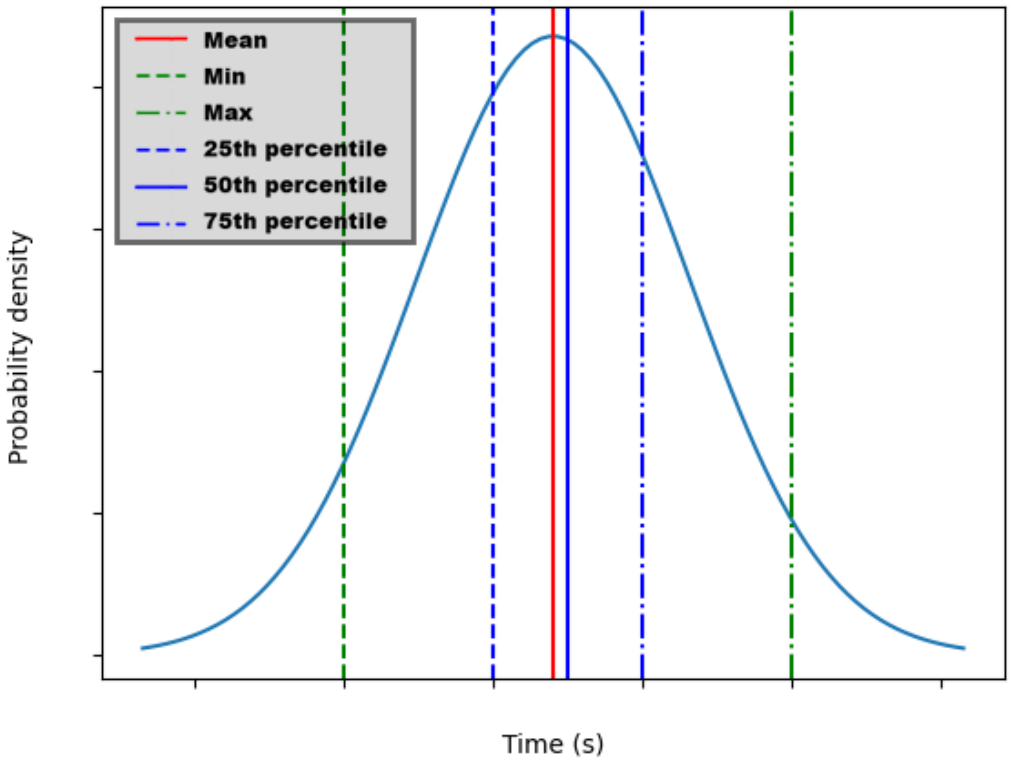
4.2 Statistik och AHP

Resultatet efter exekveringen med Data Evaluation innefattar tre normalfördelningsgrafer som visualiserar olika aspekter av systemets prestanda. De tre graferna som skapades följer utseendet av den i figur 4.5. Den första grafen visar den totala tiden för hela larmkedjan. Den andra grafen representerar tiden från att ett larm har skapats tills första bilden tagits. Den tredje grafen visar tiden från det att bilden har tagits tills dess att den registrerats i basstationen.



Figur 4.4: Visar hur den andra Raspberry Pi får signalen från den andra Raspberry Pi och därefter spelar upp vald video för Scout-en

Vidare genereras en *.csv*-fil som innehåller viktig statistisk information för att ge användaren en övergripande bild av systemets prestanda. Denna fil inkluderar mått som median, medelvärde för tidsåtgång, kvartilfördelning av tiderna, samt lägsta och högsta tidsvärden och standardavvikelse. Ett exempel kan hittas i figur 4.6 där informationen utelämnats på grund av sekretess.



Figur 4.5: Resultatet presenterades i normalfördelningskurvor som följer mallen ovan. Graferna inkluderade markörer för medel, min, max och kvantiler.

Date	List Name	Max	Min	Mean	Median	Quantiles	Standard Deviation σ	Normal Distribution
1682076993	System duration time							
1682076993	PIR alarm duration time							
1682076993	Creation to first capture time							

Figur 4.6: Ramen för CSV-filens innehåll med informationen borttagen.

4.2.1 AHP

Med det reviderade AHP-trädet i figur 3.10 kunde examensarbetarna börja bygga poängsystemet. Den nya strukturen och viktfördelningen gav en bättre förståelse för hur olika aspekter av systemet påverkade dess övergripande prestanda och poängsättningen sattes därefter. De förändringar som gjordes i AHP-trädet, såsom att öka vikten för larmmottagning och att justera kriterierna för alarmprocesseringstid och riktningsdetektering, bidrog till att tydligare framhäva de viktigaste aspekterna av systemet.

Genom att använda det reviderade AHP-trädet kunde examensarbetarna börja ställa testscenarier mot varanda och få en mer övergripande bild av systemets kontinuerliga prestanda. Detta underlättade beslutsfattandet och prioriteringar när det gällde att vidareutveckla och förbättra testningen av systemet. Resultaten från detta arbete visade att det reviderade AHP-trädet var ett effektivt verktyg för att utvärdera och poängsätta systemets prestanda.

KAPITEL 5

Slutsats

Examensarbetarna har utvecklat ett testsystem som har förmåga att testa samt utvärdera ett FLEXNET-system inomhus. Detta genom att testsystemet simulerar hela larmkedjan för systemet, samt utvärderar om larmet propagerat till basstationen. Testsytemet för larmkedjor var utformat för att ge användaren en djupare förståelse av de olika testfallens prestanda genom att på ett tydligt och informativt sätt presentera viktiga statistiska värden. Dessa värden var relaterade till tidsåtgången för de olika delarna av larmkedjan, vilket var avgörande för att bedöma systemets effektivitet och responsförmåga. Samtidigt var testsystemet mobilt och implementerat på ett sådan sätt att det finns möjlighet för att snabbt förflytta systemet runt i lokalen efter behov.

Applikationen som togs fram under arbetet var kapabel till att testa systemet. Det var utvecklat och designat för att användas av anställda på företaget och poängsättningen följer dess behov. Under utvecklingen har man i samspel med företaget implementerat företagspecifika protokoll för att kommunicera med dess enheter. Kommunikation med flera enheter har skapat ett stort behov av en stabil applikation. Långtidstesterna har bevisat att applikationen har uppfyllt kravet av att fungera under en längre period utan mänsklig påverkan.

För att garantera stabiliteten i långtidstesten har en systematisk utveckling genomförts i flera etapper. Inledningsvis utforskades metoder för att fysiskt ansluta till enheterna. Därefter genererades sensordata till sensorerna för att utvärdera om det resulterade i en korrekt larmsignal. Slutligen skickades data till flera sensorer simultant för att analysera det övergripande systemets interaktioner. När det säkerställdes att en modul fungerade korrekt blev den implementerad och integrerad med applikationen.

5.1 Svar på frågeställningar

5.1.1 Fråga 1 - Är det möjligt att använda existerande mjukvaror för att testa systemet?

Examensarbetar valde att prioritera bort denna frågeställning för att fokusera mer på resterande. Detta beror på att de andra frågeställningarna var mer relaterade till varandra och att det tog längre tid än förväntat att undersöka dem. På grund av att ingen informationsökning eller testning har gjorts i ämnet går det inte att besvara frågeställningen.

5.1.2 Fråga 2 - Hur återspelar applikationen på bästa sätt inspelad data till sensorerna?

För att spela upp data för enheter var det bästa sättet att skriva en applikation som kommunicerar med enheter enligt företagets protokoll. När det gäller PIR-en infraröda sensor, som är en väl beprövat teknik, handlade det mer om att undersöka samspelet mellan enheterna och inte om den infraröda strålningen registrerats korrekt. Därför lades ingen fokus på att återskapa en infraröd signatur, utan enbart på att skapa en digital larmsekvens.

Det bästa sättet att spela upp video för Scout-en var att använda *OpenCV* och presentera varje bild i rätt takt. För att spela upp video går det att använda Raspberry Pi vanliga bildutgång som annars visar skrivbordet. För att undvika problem med prestanda går det på ett enkelt sätt att spela upp video från en annan Raspberry Pi och endast behöva generera en startsignal från den huvudsakliga Raspberry Pi.

5.1.3 Fråga 3 - Hur skickar applikationen informationen i företagets AUX-interface?

För att skicka information i företagets aux-interface krävs det att det finns god dokumentation och kunskap om produkterna. Det tillåter modulerna att kommunicera med enheter enligt företagets protokoll. För att hantera protokollet var det smidigt att ha separata moduler som framer eller payload som översatte all information. Utöver detta har det varit tydligt att utveckla systemet när all kommunikation har skett i ett separat paket. Eftersom att i den mer abstrakta modulen som *testcase* använda paketens metoder med tydliga namn utan att behöva sätta sig in i hur kommunikationen fungerar.

En stor fördel med att ha kommunikationen separerad i ett eget paket med en driver som huvudmodul var att det förenklar felhanteringen. Om kommunikationen hade påverkats av en störning på vägen och meddelandet inte gick att läsa korrekt skulle det innebära en exception. Eftersom att den exception uppmärksammades och hanterades i paketet gjorde det att moduler som implementerade kommunikationspaketen inte behövde utveckla någon felhantering för felaktiva meddelande.

5.1.4 Fråga 6 a - Är en Raspberry Pi snabb nog för att kunna beräkna resultatet av testfallen under simulering?

Vid utveckling av applikationen noterades det att testförloppet inte upplevde prestandaproblem. Resultatet vid mätning visade att användningen av processorkraften vid testförloppet var på ungefär 5%, vilket är långt innanför marginalen för ett fungerande system.

5.1.5 Fråga 6 b - Är en Raspberry Pi kraftfull nog för att kunna simulera hela systemet?

Testningen resulterade i slutsatsen om att en Raspberry Pi är kraftfull nog att simulera hela systemet, eftersom att ungefär 87% av CPU-kraften går till att att spela upp videon. Parallellt med det går upp till ca 5% av CPU till att simulera testfall, kommunicera och beräkna. Det resultatet innebär att det går att köra hela systemet på en Raspberry Pi utan problem. Om det kommer en Raspberry Pi 5 som är bättre än 4:an kommer marginalen innan prestandaproblem att öka.

På grund av att marginalen till prestandaproblem var väldigt liten när man adderar 5 till 87, användes det två stycken Raspberry Pi under examensarbetet. Det var en enkel lösning som för priset av ytterligare en Raspberry Pi helt eliminerar funderingar över ifall den har tillräckligt med processorkraft och RAM.

Ifall varken pris eller plats är ett problem blir det enklare för användaren att separera uppgifter över flera enkortsdatorer. Då krävs det mindre kylning samtidigt som det är betydligt enklare att felsöka problem i programkoden eftersom det räcker att undersöka den enhet som har problem. Det gör det också enkelt att skala upp delar av systemet i framtiden eller att byta ut den komponent som går sönder, istället för hela systemet ifall det skulle körts på endast en enkortsdator.

5.1.6 Fråga 7 - Hur hanterar applikationen när flera enheter vill kommunicera samtidigt?

För att möjliggöra parallell kommunikation med flera enheter är det nödvändigt att varje enhet kommunicerar med en modul som körs på en egen tråd. Detta säkerställer att modulen alltid kan svara med en metod vid inkommande meddelanden. För att underlätta utvecklingen krävs det att varje modul är en klass med parametrar som konfigureras för den enheten den ska kommunicera med. I examensarbetet har två trådar använts för Scout-en och PIR-en samtidigt som den huvudsakliga kommunikationen med basstationen har skett i huvudtråden som initierat testfallen. Det garanterade att inga meddelande missades och att alla tre enheter kan skicka information *exakt* samtidigt.

Användingen av flera trådar ökar komplexiteten av programkoden och problemen som uppstår. Anledningen till det observerades under examensarbetet och var på grund av att trådar har möjligheten att arbeta förbi varandra. Om det var möjligt att göra

detta utan att försämra prestanda alltför mycket, var det mer intuitivt att utveckla moduler som exekveras sekvensiellt. Detta förbättrade kodens läsbarhet eftersom händelser alltid inträffade och exekverades i enlighet med förväntningarna. Om flera trådar används behöver utvecklaren använda sig av egenskrivna metoder eller variabler för att synka och låsa trådar vid kritiska moment.

5.2 Statistik och AHP

5.2.1 Fråga 4 - Hur presenteras statistik som på ett informativt sätt beskriver antal lyckade testsekvenser?

För att presentera statistik på ett informativt sätt använde sig examensarbetet av grafer samt statistiska mått som medelvärde, median, kvantiler och standardavvikelsen. Genom att presentera denna information i form av grafer får användaren en tydlig visualiserad bild över testresultaten. Graferna möjliggör att visuellt visa tidsåtgången för de ingående momenten i testsekvenserna, vilket ger en djupare förståelse för systemets prestanda och effektivitet. Vidare bidrar den grafiska presentationen med möjligheten att snabbt dra slutsatser som systemets prestanda, identifiera trender och mönster samt upptäcka eventuella begränsningar som kan behöva åtgärdas.

5.2.2 Fråga 5 a - Hur poängsätts ett resultat i ett testsystem som inte bara har två utfall?

Poängsättning i ett testsystem som inte bara har två utfall kan genomföras genom användandet av flerparametriska modeller som AHP-trädet. Dessa modeller tillåter att olika kriterier och aspekter av systemet värderas och poängsätts utifrån deras respektive bidrag för systemets prestanda.

5.2.3 Fråga 5 b - Vad är en lämplig skala för poängsättningen?

Vid val av skala är det viktigt att den är lämplig för det specifika systemet, samt enkel att förstå för att underlätta vid jämförelser. Lämplig skala för examensarbetet visade sig vare en intervallskala, där varje steg på skalan representerar en förändring i prestanda. I det utvecklade testsystemet används en skala från 0 till 1. Valet av skala bör grundas på systemets krav och den typ av data som samlas in för att ge en informativ och användbar bedömning av systemets prestanda.

5.2.4 Fråga 5 c - Vilka parametrar poängsätts systemet utifrån?

Parametrar som poängsätts utifrån beror på det specifika systemet som utvärderas och dess mål. I det här fallet, med ett larmsystem, används parametrar som larmmottagning, alarmprocesseringstid och riktningsdetektering då det visade sig vara mest

relevanta för att bedöma systemets prestanda. En noggrann analys av dessa parametrar ger användaren tillräckligt med information för att dra välgrundade slutsatser om systemets kapabilitet.

5.3 Reflektion över etiska aspekter

Försvarsindustrin har under de senaste åren upplevt en kraftigt ökande tillväxt på grund av det försämrade omvärldesläget, med ökade säkerhetsutmaningar och politiska spänningar. Denna tillväxt driver efterfrågan på mer avancerande och tillförlitliga system för att skydda nationella intressen samt samhällskritisk infrastruktur. I takt med ökad efterfrågan följer även högre krav på underleverantörernas produkter och tjänster. Det ställs krav på att produkter och tjänster ska uppfyller höga kvalitetsstandarder då dess arbetsuppgifter bidrar till att stärka försvarskapaciteten.

Med dessa omfattande krav på testning spelar testsystemet, som utvecklats av examensarbetarna, en väsentlig roll i att skapa samhällsnytta. Genom att erbjuda kontinuerlig testning och validering av systemets prestanda och tillförlitlighet kan underleverantörer ytterligare säkerställa att deras lösningar upprätthåller den standard som krävs för skyddandet av nationella resurser. Dessa integrerade system blir en del av det bredare försvarsförmågan, som ökar rikets förmåga att möta de växande hoten och utmaningarna i omvärlden.

5.4 Framtida utvecklingsmöjligheter

Under arbetets gång blev det tydligt att det inte var möjligt att genomföra alla tänkbara implementationer och utvecklingssteg på grund av avgränsningar i tid för projektet. Trots dessa begränsningar har mycket arbete lagts ned och viktiga framsteg har uppnåtts. Det finns fortfarande framtida visioner och områden för förbättringar i systemet som inte kunde utvecklas helt under projektets gång. Dessa framtidsvisioner och potentiella vidareutvecklingar av systemet beskrivs i följande avsnitt.

5.4.1 Testsystemet

Loggen i testsystemet har förbättringspotential. Förutom att ändra formatet för att tydligare belysa viktig information finns det även ny information som skulle varit intressant. En fördel skulle varit att logga batterinivån vid var femte test för att veta ifall saker misslyckats på grund av att enheten fick slut på batteri. Det skulle även indirekt gett användaren ett faktiskt mått på hur länge produkterna kan vara igång samt exakt vilka tester de har klarat över en viss tid på en full laddning.

En större utvecklingspotential för systemet är att ta vara på fördelarna med enkortsdatorer. Istället för att utveckla en stor applikation med flera trådar skulle en möjlighet vara att separera varje paket på en egen Raspberry Pi. Eftersom att all utveckling kan fokuseras till att testa att varje paket fungerar stabilt för sig själv. Därefter kan den

huvudsakliga Raspberry Pi kommunicera och använda de modulärna paketen för att skapa testfall och scenarion. Det leder till att applikationen blir mycket mer flexibel vid nya testfall. Nackdelen med den lösningen är att kommunikationen mellan Raspberry Pi blir något mer avancerad. Dock eftersom seriellkommunikation redan används och implementeras kan det användas mellan flera Raspberry Pi utan för mycket omskrivning.

5.4.2 Framtida möjligheter för detaljerad utvärdering av larmsystemet

Den nuvarande implementationen av AHP-träd bidrog till en god bild av prestandan baserat på de mest kritiska delarna av larmkedjan. För att ytterligare förbättra utvärderingen av larmsystemets prestanda finns det flera metoder som kan utforskas och implementeras i framtiden.

En möjlighet är att utvidga AHP-trädet för att inkludera alla delar av larmkedjan, vilket skulle bidra till en mer heltäckande bild av systemets prestanda och påverkande faktorer. Vidare skulle ett mer utvecklat AHP-träd bidra till en mer nyanserad förståelse av hur de olika aspekter i systemet interagerar och påverkar varandra.

En annan möjlighet till förbättring i AHP-trädet är att multiplicera all indata med en förändringsfaktor innan det används i uträkningarna. Detta vore bra eftersom att vissa tal anges i procent medan andra anges i sekunder, vilket gör att de bidrar olika mycket. Problemet visade sig tydligare när trädet vill fördela vikten jämnt mellan två delkriterier. Ifall 50% tas av en procentsats och 50% av en tid i sekunder, kan procentsatsens siffra vara oproportionellt stor. Till exempel om procentsatsen är 90% medan tiden är 30 sekunder blir siffran 3 gånger större. Det gör att procentsatsens faktiska påverkan blir närmare 75% medan tidens blir 25%. För att lösa det i framtiden borde procentensatsen i detta exemplet multipliceras med en förändringsfaktor på 0.33 för att ge mer sanningsenliga resultat.

Förutom att utveckla AHP-trädet är det önskvärt att andra statistiska metoder används, dels för att fördjupa analysen av larmsystemets prestanda, samt att ha möjlighet att korsvalidera sina statistiska modeller. Exempelvis kan regressionsanalys användas för att modellera sambandet mellan en beroende variabel, såsom prestanda eller tidsåtgång, och en eller flera oberoende variabler, såsom sensortyper och miljöförhållanden. Detta kan hjälpa till att förstå hur olika faktorer påverkar larmsystemets prestanda och förutse hur förändringar i dessa faktorer kan påverka framtida prestanda.

Genom att implementera dessa avancerade statistiska metoder kan larmsystemets prestanda utvärderas mer detaljerat, vilket leder till mer omfattande testning och därmed säkerställandet av systemets optimering för framtida behov samt utmaningar.

5.4.3 Vision

För att vidareutveckla systemet och optimera dess användning föreslår examensarbetarna att systemet implementeras för systematisk testning under hela utvecklingspro-

cessen. Genom att utföra slumpmässiga och löpande tester på enheterna under alla delar av utvecklingsprocessen kan företaget säkerställa att olika aspekter av systemet kontinuerligt utvärderas och förbättras. Vidare bidrar dessa löpande tester till att ge företaget en solid grund för att förstå systemets normala funktion och poäng. Detta underlättar jämförelser mellan olika system och bedömningar av hur prestandan påverkas av olika inställningar och modifikationer. Genom att etablera en baslinje av testdata kan företaget på ett effektivt sätt upptäcka och analysera eventuella avvikelser i prestanda och utvärdera hur de relaterar till specifika justeringar i systemet.

KAPITEL 6

Terminologi

.csv – Comma-separated values är ett textfilsformat som används för att spara och överföra tabelldata.

Dictionary – En datatyp i Python som lagrar nyckel-värde par.

Exception – Ett fel som uppstår när programmet körs som leder till att det avbryts om det inte hanteras i exempelvis ett try-except-block.

Float – En datatyp som används i Python för att ange decimaltal, eller även kallat flyttal.

IDE – Integrated Development Environment är en programvara som kombinerar flera verktyg för att skriva och felsöka kod.

Jitter – Ett begrepp för resultatet av en störning eller variation av förändring i ett kommunikationssystem.

NaN – Ett speciellt värde som ofta används inom numeriska beräkningar. Det representerar ett värde som inte är definierat eller kan representeras som ett giltigt tal.

None – Ett värdet som används i Python för att ange avsaknad av värde, liknande *null* i Java.

None-type – En datatyp i Python som representerar avsaknad av värde.

PIR – En förkortning för Passiv Infraröd Sensor. I företagets system kallas vanligtvis den sensor som är kopplad till baskroppen för ett PIR-rör. Produkten är tvådelad och består av en sensor(PIR-rör) och en kropp(Meshbas). Vid benämning i rapporten beskriver ordet PIR istället kroppen(Meshbas) som Raspberry Pi är kopplad till.

Timing – Ett begrepp som beskriver tidpunkten eller fördröjningen som är kritiskt. Används ofta för att beskriva samverkan med hjälp av fördröjningar.

While-loop – Ett kodblock inom programmering som upprepar sitt innehåll så länge dess villkor är uppfyllt.

Litteratur

[1] Python Software Foundation. *Python FAQ - General*. 2023. URL: https://docs.python.org/3/faq/general.html#what-is-python (hämtad 2023-04-18).

[2] Python Software Foundation. *time — Time access and conversions*. 2023. URL: https://docs.python.org/3/library/time.html (hämtad 2023-03-30).

[3] JSON.org. *JSON: The Fat-Free Alternative to XML*. 2023. URL: https://www.json.org/json-en.html (hämtad 2023-04-19).

[4] Python Software Foundation. *json — JSON encoder and decoder*. 2023. URL: https://docs.python.org/3/library/json.html (hämtad 2023-03-30).

[5] Hunter John, Dale Darren, Firing Eric, Droettboom Michael och the Matplotlib development team. *Matplotlib: A 2D Graphics Environment*. 2007. URL: https://matplotlib.org/stable/contents.html (hämtad 2023-03-30).

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Martin Haldane, Jaime Fernández del Río Beojan, Marvin Wiebe, Paul Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke och Travis E. Oliphant. *Array programming with NumPy*. 2020. URL: https://numpy.org/doc/stable/index.html.

[7] Python Software Foundation. *statistics — Mathematical statistics functions*. 2023. URL: https://docs.python.org/3/library/statistics.html (hämtad 2023-03-30).

[8] Sparkfun Electronics. *Serial Communication*. 2012. URL: https://learn.sparkfun.com/tutorials/serial-communication (hämtad 2023-02-27).

[9] Raspberry Pi. *Raspberry Pi hardware*. 2023. URL: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (hämtad 2023-02-27).

[10] R.W. Saaty. 'The analytic hierarchy process—what it is and how it is used'. I: *Mathematical Modelling* 9.3 (1987), s. 161–176. ISSN: 0270-0255. DOI: https://doi.org/10.1016/0270-0255(87)90473-8. URL: https://www.sciencedirect.com/science/article/pii/0270025587904738.

[11] Exensor. *Flexnet - Flexible Network of Sensors*. 2023. URL: https://www.exensor.com/products/flexnet-flexible-network-of-sensors/ (hämtad 2023-04-24).

[12] Exensor. *Gateway - Flexnet System Network*. 2023. URL: https://www.exensor.com/products/flexnet-flexible-network-of-sensors/system-network/gateway/ (hämtad 2023-04-24).

[13] Exensor. *PIR - Unattended Ground Sensors (UGS) - Flexnet*. 2023. URL: https://www.exensor.com/products/flexnet-flexible-network-of-sensors/unattended-ground-sensors-ugs/pir/ (hämtad 2023-04-24).

[14] Exensor. *Scout Mk3 - Electro-Optics - Flexnet*. 2023. URL: https://www.exensor.com/products/flexnet-flexible-network-of-sensors/electro-optics/scout-mk3/ (hämtad 2023-04-24).

[15] Jose Vilches. 'Interview with Raspberry's Founder Eben Upton'. I: (2012). URL: https://www.techspot.com/article/531-eben-upton-interview/ (hämtad 2023-04-19).

[16] Raspberry Pi. *Raspberry Pi 4*. 2023. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (hämtad 2023-05-02).

[17] Electrokit Sweden AB. *Raspberry Pi 4 Model B/4GB*. 2023. URL: https://www.electrokit.com/produkt/raspberry-pi-4-model-b-4gb/ (hämtad 2023-05-02).

[18] Electrokit Sweden AB. *Raspberry Pi 3 1GB model B+*. 2023. URL: https://www.electrokit.com/produkt/raspberry-pi-3-1gb-model-b-2/ (hämtad 2023-05-02).

[19] Guido van Rossum, Barry Warsaw och Nick Coghlan. *PEP 8 – Style Guide for Python Code*. 2001. URL: https://peps.python.org/pep-0008/ (hämtad 2023-04-26).

[20] Inc Electronic Team. *RS485 communication guide*. 2021. URL: https://www.eltima.com/article/rs485-communication-guide/ (hämtad 2023-03-30).

[21] Paul Kirvan. *Network Time Protocol (NTP)*. 2023. URL: https://www.techtarget.com/searchnetworking/definition/Network-Time-Protocol (hämtad 2023-04-25).

[22] Zeng Xuyun. 'What programming language should you use with the Raspberry Pi?' I: (2022). URL: https://picockpit.com/raspberry-pi/what-programming-language-should-you-use-with-the-raspberry-pi/ (hämtad 2023-04-19).

[23] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009. ISBN: 978-0-13-235088-4.