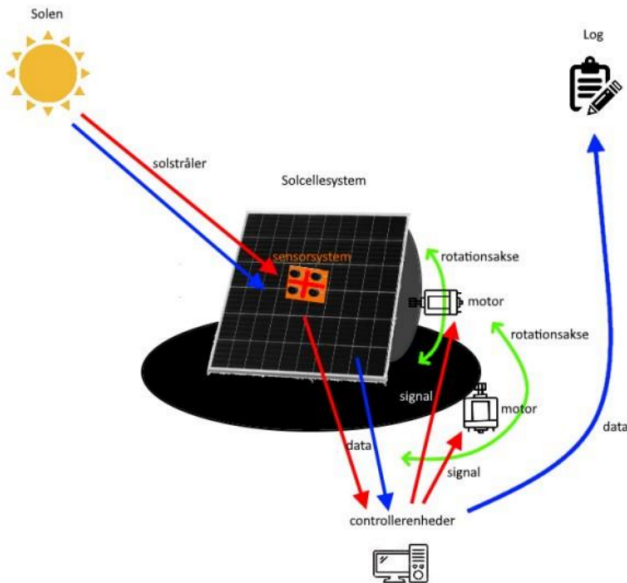


IoT Based Dual-Axis Solar Tracking System

Gruppe 9

13. december 2024



Navn	Studienummer
Luqman Abdirisak Ali Nur	202304925
Mikail Dogan Cetinalp	202307879
Ifa Moti Deressa Gutema	202108255
Bleron Gashi	202307840
Karam Alchamat	202307313
Jahye Ali Hussen	202309135
Kevin Pham	202308485

$$\text{Antal tegn}(\text{med mellemrum}) = 61483$$

1 Resume

Projektet, IoT-baseret Dual-Axis Solar Tracking System (IoT-DASTS), er udviklet med det mål at maksimere solenergiproduktion ved at forbedre solpanelernes effektivitet gennem avanceret teknologi. Traditionelle solcelleanlæg har ofte begrænsninger, da de ikke dynamisk justerer deres position i forhold til solens bevægelser, hvilket fører til suboptimal energiproduktion. IoT-DASTS adresserer dette ved at integrere dynamisk sporing, realtidsdataindsamling og et brugervenligt webbaseret dashboard.

Systemet bruger lyssensorer til at registrere solens position og justerer solpanelerne i både horisontal og vertikal retning ved hjælp af stepper- og servomotorer. Dette sikrer, at panelerne konstant er vinklet optimalt for maksimal eksponering mod solen. Samtidig indsamler systemet miljødata som lysintensitet, temperatur og luftfugtighed, hvilket ikke kun giver brugeren indsigt i energiproduktionen, men også muliggør bedre styring og analyse af anlæggets effektivitet.

Projektet er udviklet med Scrum som projektstyringsmetode og bygger på komponenter som Raspberry Pi, ESP32 og PT550-lyssensorer. Test af nøglefunktioner har vist succes, og systemet lever op til de fleste krav om funktionalitet og brugervenlighed.

Fremadrettet kan IoT-DASTS forbedres med endnu bedre energieffektivitet, integration af yderligere sensorer og cloud-baseret dataanalyse.

2 Abstract

The IoT-based Dual-Axis Solar Tracking System (IoT-DASTS) was developed with the goal of maximizing solar energy production by enhancing the efficiency of solar panels through advanced technology. Traditional solar panel systems often have limitations, as they do not dynamically adjust their position to follow the sun's movements, leading to suboptimal energy output. IoT-DASTS addresses these issues by integrating dynamic tracking, real-time data collection, and a user-friendly web-based dashboard.

The system utilizes light sensors to detect the sun's position and adjusts the solar panels both horizontally and vertically using stepper and servo motors. This ensures that the panels are continuously angled for optimal sun exposure. Simultaneously, the system collects environmental data such as light intensity, temperature, and humidity, providing users with insights into energy production and enabling better control and analysis of the system's performance.

The project was developed using the Scrum methodology and is built on components such as Raspberry Pi, ESP32, and PT550 light sensors. Tests of key functionalities demonstrated success, and the system meets most requirements for functionality and user accessibility.

Future enhancements for IoT-DASTS include improved energy efficiency, integration of additional sensors, and cloud-based data analytics.

3 Forord

Denne rapport er udarbejdet af en gruppe studerende på Aarhus Universitet Elektro- og computerteknologi. Gruppen består af: Luqman Abdirisak Ali Nur, Mikail Dogan Cetinalp, Ifa Moti Deressa Gutema, Bleron Gashi, Karam Alchamat, Jahye Ali Hussen og Kevin Pham, som er blevet vejledt af Lars Mortensen.

Rapporten er resultatet af semesterprojektet på 3. semester. Rapporten er udarbejdet i overensstemmelse med de læringsmål, der er fastlagt af Aarhus Universitet for dette semesterprojekt. Det skriftlige arbejde afleveres den 13. december 2024.

4 Definitionsliste

Forkortelse	Oprindelige ord
IBD	Internal Block Diagram
BDD	Block definition diagram
UC	Use case
SD	Sekvensdiagram
FURPS+	Funcionaliet, Useability, Reliability, Performance, Supportability
MoSCoW	Must have, Should have, Could have and Will not have
ESP	Electronic Stability Program
RPI	Raspberry pi
IoT-DASTS	IoT Based Dual-Axis Solar Tracking System
TDD	Test-driven development
ASE	ASE modellen

5 Arbejdsfordeling

Ansvarsområde	Kevin	Jahya	Ifa	Bleron	Luqman	Mikail	Karam
Implementering motor controller	O	X			O	X	O
Implementering lyssensor	O	O	X	X			
Implementering strømsensor	X				X		
Implementering <u>HTU21</u>	X	O			X		X
Implementering ESP32		X					O
Implementering Raspberry Pi	O	X				X	X
Implementering webclient		O	X	X			

Figure 1: Arbejdsfordelings tabel

Contents

1	Resume	2
2	Abstract	2
3	Forord	2
4	Definitionsliste	3
5	Arbejdsfordeling	3
6	Indledning	5
6.1	Projektkilde	5
6.2	Problemformulering	5
6.3	Ressourcer og projektafgrænsning	6

7	Metoder og arbejdsproces	7
7.1	ASE Modellen	7
7.2	Project management (Scrum)	7
7.3	Agil udvikling	9
8	Krav og prioriteringer	9
8.1	Rammekrav til projektet	9
8.1.1	Aktør kontekst diagram	9
8.1.2	Aktør beskrivelse	10
8.2	Funktionelle krav specificeret vha. use cases	10
8.2.1	Use case diagram	10
8.2.2	Fully case beskrivelse	11
8.3	Ikke-funktionelle krav	15
8.3.1	FURRPS+	15
8.4	Prioritering af funktionalitet(vha. MoSCoW)	16
8.4.1	MoSCoW	16
9	Indledende analyser	16
9.1	Teknologianalyse	16
9.1.1	Hardware	16
9.1.2	Software	19
9.2	Risikoanalyse	20
9.2.1	Projekt omfang	20
9.2.2	Tidspres	20
9.2.3	Management	21
9.2.4	Teamwork/Samarbejde	21
9.2.5	Frafald eller fravær	21
10	Systemarkitektur	21
10.1	Domænenemodell	21
10.2	BDD'er og IBD	22
10.2.1	Block Definition Diagram	22
10.2.2	Internal Block Diagram	23
10.3	Sekvensdiagrammer for handling	24
10.3.1	Sekvensdiagram for UC1	24
10.3.2	Sekvensdiagram for UC2	26
11	Modul- og kommunikation	28
11.1	Hardware og software design	28
11.2	Hardware design	28
11.3	Software design	28
11.3.1	Flowchart for UC1	28
11.3.2	Applikationsmodel baseret på UC1 og UC2	29
12	Realisering af systemet	33
12.1	Hardware implementering	33
12.2	Software implementering	34
12.2.1	Implementering af webclien t	34
12.2.2	1. Server-Side: Hndter Temperaturforesprgsler	34
12.2.3	Implementering af motor	34
12.2.4	Implementering af HTU21	36
12.2.5	Implementering af lyssensor	36

13 Tests og resultater	39
13.1 Modultest	39
13.2 Accepttest	44
13.2.1 Accepttest for UC1 - Bevægelse	44
13.2.2 Accepttest for UC2 - Strømgenering	44
13.3 Hovedresultater	44
14 Diskussion af resultater	45
15 Konklusion og perspektivering	45
16 Bilagsliste	46
17 Referenceliste	46

6 Indledning

6.1 Projektkilde

Projektet er centreret omkring udviklingen af et IoT-baseret Dual-Axis Solar Tracking System, som optimerer solenergiproduktion ved at justere solpanelernes orientering dynamisk i realtid.

Baggrund og motivation: Solenergi er en bæredygtig energikilde, men effektiviteten afhænger stærkt af solpanelernes orientering mod solen. Eksisterende systemer mangler ofte dynamisk tilpasning til solens bevægelser og indsamling af præcise miljødata. Dette projekt søger at løse disse problemer ved at kombinere avanceret sensortechnologi, realtidsdataindsamling og motorstyring med IoT-løsninger.

At udvikle et intelligent system, der:

1. Dynamisk tilpasser solpanelernes position ved hjælp af lys- og vejrsensorer.
2. Overvåger og logger miljødata som temperatur, luftfugtighed og lysintensitet.
3. Giver brugerne adgang til energiproduktion og miljødata via et webbaseret dashboard.

6.2 Problemformulering

Effektiv udnyttelse af solenergi afhænger i høj grad af solcellepanelers orientering mod solen og de aktuelle vejrforhold. Mange eksisterende systemer mangler dynamisk tilpasning til solens bevægelser og realtidsindsamling af miljødata, hvilket resulterer i suboptimal energiproduktion. Derudover er brugervenlig monitorering og strømstyring ofte fraværende, hvilket gør det vanskeligt for brugerne at overvåge og optimere deres solcelleanlæg.

For at løse dette problem er der behov for et intelligent IoT-baseret system, der kan overvåge solens position og miljødata i realtid, justere solpanelernes orientering dynamisk og levere strøm effektivt. Systemet skal desuden give brugerne adgang til detaljerede målinger og produktionsdata via et dashboard for at sikre gennemsigtighed og kontrol over energiproduktionen.

Målet med projektet er derfor at udvikle og implementere et automatiseret system. Systemet skal være i stand til dynamisk at justere solpanelernes orientering ved hjælp af en toakset soltracker og lys-sensorer for at opnå maksimal energiproduktion. Derudover skal det logge miljødata som temperatur og luftfugtighed i realtid, hvor disse data gemmes både lokalt og i en cloud-løsning med henblik på videre analyse. For at give brugerne nem adgang til information skal systemet overvåge energiproduktion og miljødata via et webbaseret dashboard, der løbende opdateres med

ny information. Endelig skal systemet sikre stabil og effektiv strømproduktion, som kan leveres til batterier, elnettet eller direkte til tilsluttede apparater.

6.3 Ressourcer og projektafgrænsning

1. **Lyssensor:** Lyssensoren bruges til at måle solens lysintensitet og bestemme dens position på himlen. Denne sensor leverer data, som hjælper systemet med at justere solpanelernes orientering i forhold til solens bevægelse i løbet af dagen.
2. **Temperatur og fugtighedssensor:** Denne sensor overvåger miljøforholdene ved at måle temperatur og luftfugtighed i realtid.
3. **Strømsensor:** Strømsensoren måler den strøm, som genereres af solpanelerne, og overvåger energiforbruget i systemet.
4. **Raspberry-pi:** Raspberry Pi fungerer som hovedcontrolleren for systemet og håndterer motorstyringen. Den modtager input fra lyssensorerne for at beregne solens position og sender styringssignaler til motorerne, så solpanelerne kan justeres i både horisontal og vertikal retning. Raspberry Pi sikrer også synkronisering mellem de forskellige systemdele og muliggør eksekvering af avancerede algoritmer til solsporing og optimering.
5. **ESP32:** ESP32-enheden er ansvarlig for at indsamle data fra temperatur- og fugtighedssensorer samt lyssensorer. Den fungerer som en bro mellem systemet og serveren ved at transmittere realtidsdata til skyen eller en lokal server via Wi-Fi. ESP32 kan også opdatere det webbaserede dashboard med vejrinformation og produktionsdata, så brugerne altid har adgang til opdateret information om deres
6. **Display:** Displayet gav brugerne mulighed for at overvåge systemets data, såsom temperatur, luftfugtighed, lysintensitet og energiproduktion, direkte på enheden
7. **Motorcontroller (stepper- og servo):** Motorerne muliggør rotation og tilt af solpanelerne, så de kan justeres i to akser (horisontal og vertikal) for at følge solens bevægelser og maksimere energiproduktionen.
8. **Solcelle:** Solcellen er kernen i systemet og genererer elektricitet fra solens energi. Den spiller en afgørende rolle i systemets effektivitet og tilpasses dynamisk ved hjælp af motorernes bevægelse for at opnå den optimale solindfaldsvinkel.

7 Metoder og arbejdsproces

7.1 ASE Modellen



Figure 2: ASE MODEL

Udviklingsprocessen for IoT-Based Dual-Axis Solar Tracking System er struktureret ved hjælp af ASE-modellen, som vi tidligere blev introduceret til i forbindelse med Semesterprojekt 1 og 2 på Software-uddannelsen. ASE-modellen er en central metode, der sikrer, at projektarbejdet indeholder vigtige faglige elementer og danner et struktureret helhedsbillede af systemet.

Arbejdet startede med udarbejdelsen af grundlæggende projektdokumenter såsom projektformulering, kravspecifikation og arkitektur. Dette skabte et solidt fundament for den iterative proces, hvor design og implementering af hardware- og softwaremoduler blev udført. Undervejs har vi gennemført flere iterationer af test for at validere og forbedre systemets kvalitet.

ASE-modellen tilbyder en iterativ og fleksibel tilgang, der gør det muligt at håndtere komplekse tekniske opgaver på en systematisk måde. For at styrke arbejdsprocessen har vi suppleret modellen med Scrum-metoden, som hjalp os med at organisere opgaverne og opdele arbejdet i mindre, håndterbare dele. Scrum bidrog til at sikre kontinuerlige leverancer og fremmede effektivt samarbejde i projektgruppen.

Som afslutning på projektet har vi udført integrationstest og accepttest for at sikre, at systemet lever op til de opstillede krav og forventninger. Kombineret har ASE-modellen og Scrum-metoden givet os en robust ramme til at strukturere og gennemføre udviklingsprocessen effektivt og målrettet.

7.2 Project management (Scrum)

I projektet anvendte vi en kombination af Scrum og tekniske metoder for at sikre en struktureret og fleksibel arbejdsproces, der kunne håndtere både kravændringer og tekniske udfordringer. Fra projektets start blev målene og arbejdsopgaverne nøje planlagt gennem en fælles workshop, hvor vi etablerede en produktbacklog(monday). Backloggen fungerede som en dynamisk liste over alle

nødvendige opgaver, prioriteret efter betydning og kompleksitet. Vi fordelte roller i gruppen, så ansvaret blev tydeligt, og vi kunne arbejde effektivt i mindre teams på tværs af hardware, software og integration.

Scrum var den overordnede projektstyringsmetode, som vi tilpassede vores behov. Projektet blev opdelt i korte, iterative sprints, hvor vi fokuserede på at levere små, funktionelle dele af systemet. Hvert sprint startede med en planlægningsfase, hvor vi udvalgte opgaver fra backloggen og satte klare mål. I løbet af sprintene havde vi ugentlige møder, hvor vi delte status, udfordringer og næste skridt. Dette gjorde det muligt at opdage og løse problemer hurtigt.

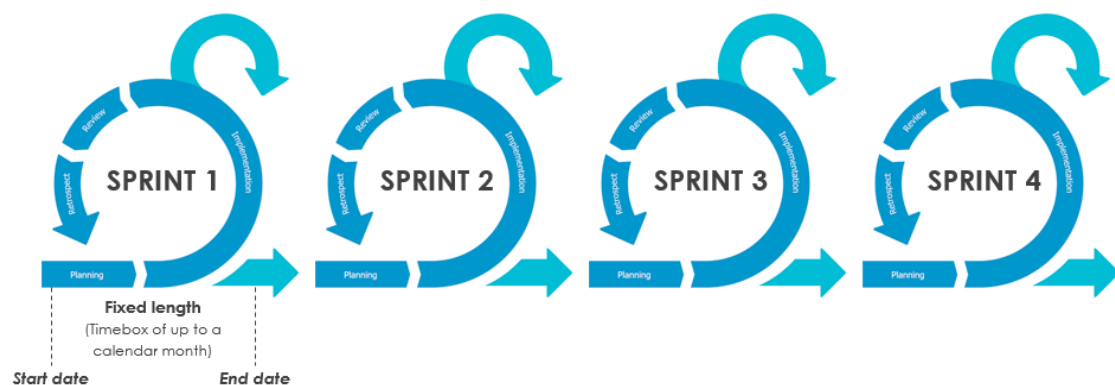


Figure 3: Scrum

Udviklingen blev understøttet af tekniske metoder som prototyping og testdreven udvikling (TDD). Vi skabte tidlige prototyper af både hardware og software for at validere vores designvalg og reducere risikoen for fejl senere i projektet. TDD blev især anvendt til integrationen af sensorer, hvor vi sikrede, at funktionaliteten blev testet og dokumenteret løbende. Dokumentation var generelt en integreret del af processen og hjalp os med at dele viden i gruppen og holde interesser opdaterede.

Integration og test var en kontinuerlig del af arbejdsprocessen. Hardware- og softwarekomponenterne blev løbende integreret og testet for at sikre, at alle dele fungerede sammen. For eksempel testede vi præcisionen af servo- og steppermotorer for at sikre korrekt solsporing, og vi verificerede sensorernes nøjagtighed ved små eksperimenter under forskellige forhold. Feedback fra vejleder og andre grupper spillede en central rolle og blev indarbejdet for at forbedre både funktionalitet og brugervenlighed, som da vi reviderede designet af webgrænsefladen baseret på tidlige tilbagemeldinger.

SEMESTER PROJEKT 3

Integrate Automate Invisio 1 69

Main Table New Item Search Person Filter Sort Hide Group by

> Dagsorden	Status	Ch...	Assigned		
7 Items / 24 SubItems	0/77				
> Rapport skrivning indhold	Status	Ch...	Assigned		
16 Items / 29 SubItems	2/16				
Sprint 1 - Problem Formulering (introduktion)					
Item	Status	Ch...	Assigned	Text	
Problemformulering	Finished	✓			
> Krav og prioriteringer	Finished	✓			
> Risiko Analyse	Finished	✓		Mangler et skrevet afsnit omkring vores risiko analyse	
> Teknologianalyse	Finished	✓		Mangler analyse for Aktuator	
4/4					
> Sprint 2 - Use Case 1 (Sun Tracking)	Status	Ch...	Assigned		
4 Items / 15 SubItems	2/4				
> Sprint 3 - Use Case 2 + 3 (Data beh...	Status	Ch...	Assigned		
6 Items / 11 SubItems	0/6				
> Sprint 4 - Use Case 4 (Strømgeneri...	Status	Ch...	Assigned		
5 Items / 9 SubItems	0/5				
> Sprint 5 - Test	Status	Ch...	Assigned		
4 Items	0/4				
> Sprint 6 - Conclusion	Status	Ch...	Assigned		
2 Items / 6 SubItems	0/2				

Figure 4: Monday

7.3 Agil udvikling

Projektet følger en agil udviklingsmetodik for at kunne tilpasse sig ændrede krav og udfordringer undervejs. Agil udvikling prioriterer små, hyppige leverancer frem for store, komplekse leverancer, hvilket reducerer risikoen for forsinkelser og tekniske problemer. Ved at levere delmål løbende kan feedback fra interessenter hurtigt integreres i det videre arbejde.

Samarbejdet i teamet er tæt og dynamisk, hvilket giver mulighed for at udnytte individuelle styrker og fremme innovation. Iterativ test og evaluering af løsninger er integreret i processen for at sikre, at systemet opfylder kravene og fungerer optimalt.

8 Krav og prioriteringer

Dette afsnit indeholder systemspecifikationen for den fulde version af IoT-DASTS. Systemet beskrives ved hjælp af et aktør-kontekstdiagram og detaljerede Fully Dressed Use Case-beskrivelser. Derudover præsenteres og analyseres IoT-DASTS' funktionelle og ikke-funktionelle krav ved brug af MoSCoW- og FURPS-analysemodellerne baseret på systemet.

8.1 Rammekrav til projektet

For at sikre, at projektet kan gennemføres med succes, er der fastlagt klare rammekrav, der dækker både tekniske, funktionelle og organisatoriske aspekter.

8.1.1 Aktør kontekst diagram

Vi benytter en aktør-kontekstdiagram, da det giver en visuel repræsentation af systemets interaktion med de involverede aktører. Dette hjælper med at forstå, hvordan systemet fungerer i forhold til både primære og sekundære aktører.

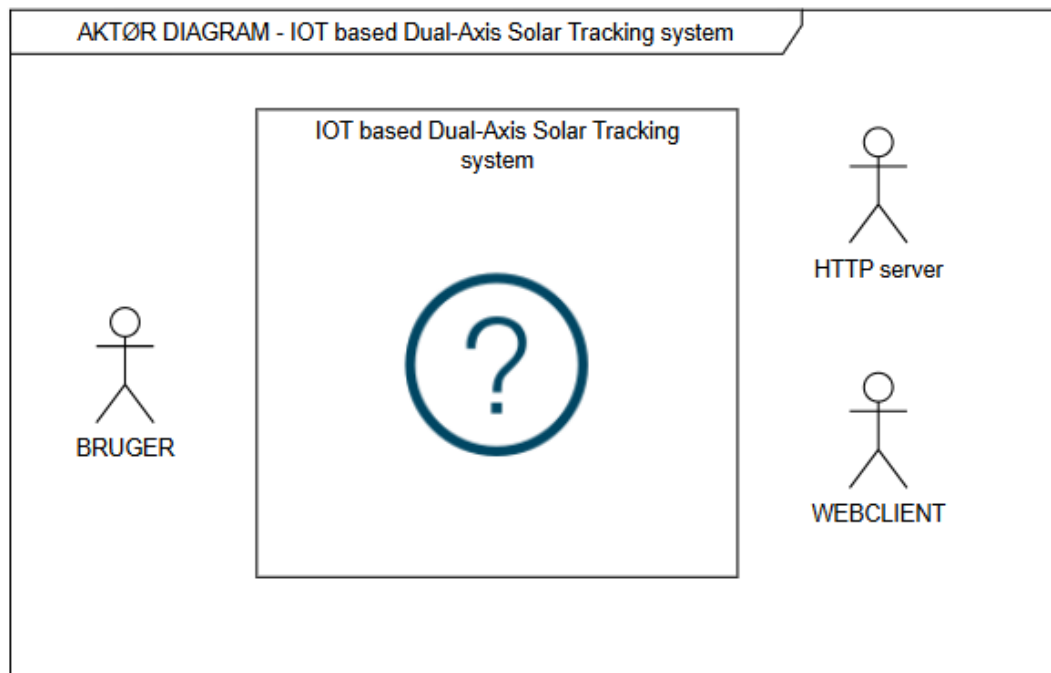


Figure 5: Aktør kontekst diagram for det IoT-baserede Dual-Axis Solar Tracking System

8.1.2 Aktør beskrivelse

Primær aktør:

Aktør type	Beskrivelse	Interaktion med systemet
Bruger	Den menneskelige aktør, som opstarter systemet og interagerer med webklienten. Bruger kan overvåge og modtage data fra systemet.	Brugeren får adgang til realtidsdata via et webbaseret website. Her kan de overvåge vejrforhold og systemets effektivitet.

Sekundær aktør:

Aktør type	Beskrivelse	Interaktion med systemet
HTTP SERVER	Repræsenterer den protokol, der bruges til at kommunikere med systemet via nettet. Bruges til at modtage data mellem IoT-systemet.	HTTP-serveren fungerer som en bro mellem sensordata og webklienten ved at håndtere forespørgsler og videresende data.
Webclient	En ekstern applikation eller enhed, der interagerer med IoT-systemet via internettet. Den fungerer som en grænseflade for eksterne brugere	Webklienten trækker data fra HTTP-serveren og præsenterer dem i et brugervenligt dashboard

8.2 Funktionelle krav specificeret vha. use cases

8.2.1 Use case diagram

Dette use case-diagram repræsenterer det IoT-DASTS, som er designet til at optimere solenergiproduktionen ved dynamisk at justere solpanelernes orientering baseret på realtidsdata.

Systemet integrerer avanceret IoT-teknologi til at udføre nøglefunktioner såsom bevægelseskontrol, dataindsamling, realtidsmonitorering og strømgenerering. Hver funktion er repræsenteret som en use case med relevante aktører, der interagerer med systemet.

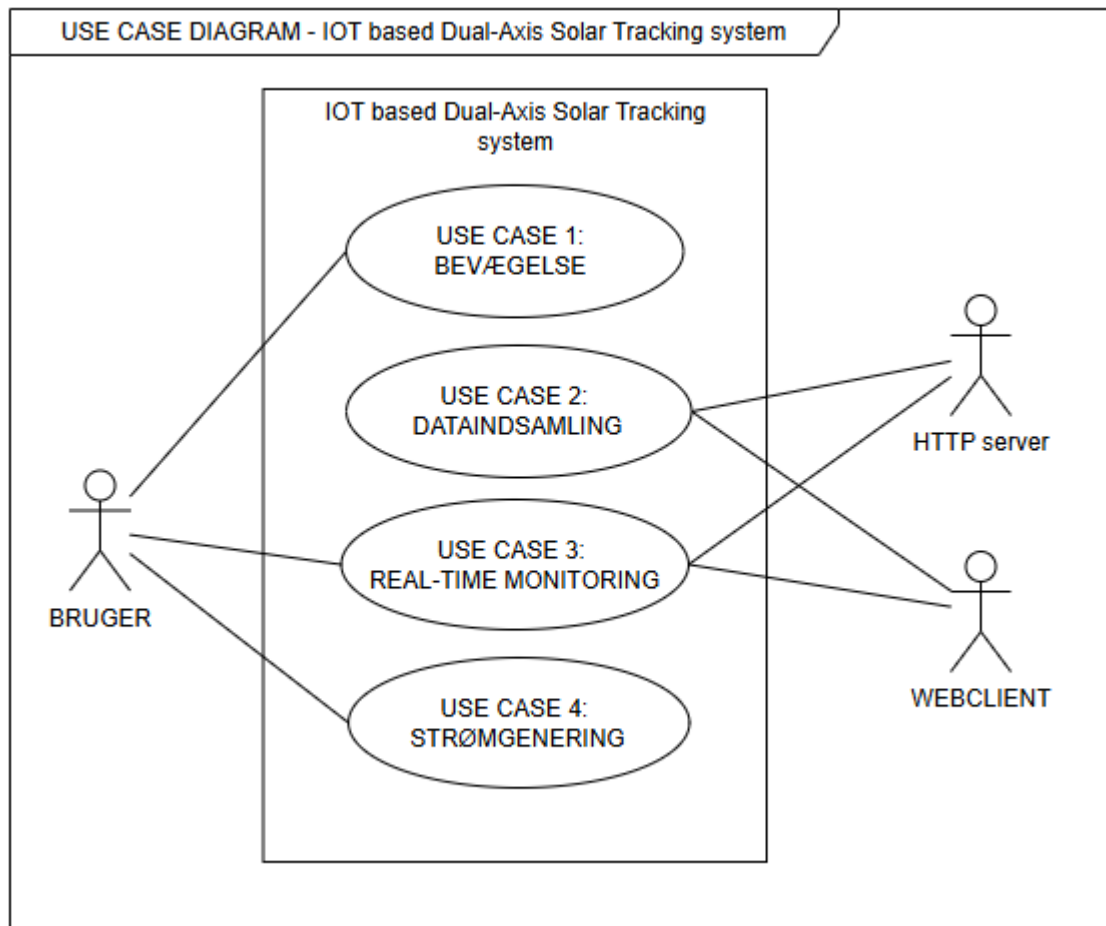


Figure 6: Use case-diagram for det IoT-baserede Dual-Axis Solar Tracking System

8.2.2 Fully case beskrivelse

Dette afsnit beskriver IoT-DASTS' Use Cases ved hjælp af fully dressed Use Case-beskrivelser. Formålet med denne tilgang er at skabe en detaljeret beskrivelse af systemet og identificere potentielle mangler. Det bemærkes, at de fully dressed Use Case-beskrivelser repræsenterer de funktioner, der oprindeligt var tiltænkt IoT-DASTS, inden beslutningen om at implementere en beta-version blev truffet.

Rapporten fokuserer primært på UC1 og UC2, da disse Use Cases indeholder de mest essentielle funktionaliteter for IoT-DASTS. Derfor vil kun to fully dressed Use Case-beskrivelser for hver af disse cases blive præsenteret her. De øvrige fully dressed Use Case-beskrivelser kan findes i bilagsrapporten under afsnittet om kravspecifikation.

USE CASE 1 - Bevægelse

Navn	Bevægelse
Mål	At justere solpanelernes orientering dynamisk for at optimere eksponeringen mod solen og maksimere energioptagelsen.
Initiering	Brugeren starter systemet og processen modtager data fra lyssensorerne eller efter en fastlagt tidsperiode (f.eks. hvert 10. sekund).
Aktør	Bruger (Primær)
Antal samtidige forekomster	1
Prækondition	Systemet er aktiv og alle komponenter virker.
Postkondition	Solpanelerne er justeret til den mest optimale vinkel mod solen, baseret på lysintensiteter. Og er i standby.
Hovedscenarie	<ol style="list-style-type: none"> 1. Lyssensorer registrerer lysintensitet i fire retninger: Venstre, Højre, Op, Ned. 2. ESP32 videregiver disse data via UART til Raspberry Pi. 3. Systemet identificerer retningen med den højeste lysintensitet. 4. Raspberry Pi læser data fra den serielle port. 5. Raspberry Pi bruger PID-controlleren til at beregne, hvor meget motorerne skal justere: <ul style="list-style-type: none"> • Stepmotoren justeres horisontalt (Venstre eller Højre). • Servomotoren justeres vertikalt (Op eller Ned). 6. Raspberry Pi sender kontrolsignaler til motorerne via GPIO. 7. Stepmotoren drejer det beregnede antal trin mod venstre eller højre. 8. Servomotoren bevæger sig til den beregnede vinkel op eller ned. 9. Raspberry Pi overvåger motorernes position og sikrer, at solpanelerne når den ønskede vinkel. 10. Hvis positionen er korrekt justeret, logger systemet resultatet. 11. Systemet går i standby og venter på næste dataindsamling (hver 10 sekunder eller en tilsvarende periode).
Udvidelser (undtagelser)	<p>EXT 1</p> <ol style="list-style-type: none"> 1. Motoren er allerede indenfor en acceptabel fejlmargen i forhold til den beregnede optimale vinkel 2. Systemet springer motoraktiveringen over og går direkte i standby.

Table 1: Use case: Bevægelse

USE CASE 2 - Dataindsamling

Navn:	Dataindsamling
Mål:	At måle temperatur, luftfugtighed og lysintensitet, behandle data, sende data til webklienten og præsentere dem lokalt.
Initiering:	Systemet starter processen ved at indsamle data fra HTU21-sensoren og lyssensoren.
Aktør:	<ul style="list-style-type: none">• HTTP Server (Sekundær)• Webklient (Sekundær)
Antal samtidige forekomster:	Ingen
Prækondition:	<ul style="list-style-type: none">• Systemet er aktivt og fungerer korrekt.• HTU21-sensoren og lyssensoren virker.
Postkondition:	<ul style="list-style-type: none">• Data er sendt til webklienten.• Dataene er vist lokalt.
Hovedscenarie:	<ol style="list-style-type: none">1. ESP anmoder om data fra HTU-sensoren.2. Lyssensorernes data aflæses via analoge input (ADC).3. ESP kontrollerer/validerer, at data er inden for gyldige intervaller. EXT 14. Rådata konverteres til det rigtige format.5. Behandlede data præsenteres lokalt via TFT-skærmen.6. Behandlede data sendes til serveren.7. HTTP-serveren gemmer midlertidigt dataen.8. Serveren sender data til webklienten via HTTP POST-anmodning.9. Systemet opdaterer data kontinuerligt (fx hvert minut eller efter brugerens præference).
Udvidelser (udtagelser):	EXT 1: <ol style="list-style-type: none">1. Hvis ingen ændringer i data opdages:2. Spring ESP-opdatering af displayet og HTTP-overførslen over.

USE CASE 3 - Real-time Monitoring

Use casen giver brugerne mulighed for at overvåge solpanelets præstation og miljødata i realtid via en webbaseret webclient. Processen starter, når brugeren åbner webklienten gennem en web-

browser ved hjælp af ESP's IP-adresse eller aktiverer det lokale display. Systemet er bygget omkring tre aktører: brugeren, som interagerer med webclienten; HTTP-serveren, der behandler og leverer data; og webbrowseren, der viser webclienten med grafer og tabeller.

USE CASE 4 - Strømgenerering

processen med at generere elektricitet ved hjælp af solpaneler og give en indikation af strømproduktionen. Strømgenereringen starter automatisk, når sensorer registrerer sollys, eller manuelt, hvis brugeren aktiverer systemet. Forudsætningerne for korrekt funktion inkluderer, at solpanelerne er korrekt installeret, sensorer og systemkomponenter fungerer optimalt, og at batterienheden eller tilsluttede apparater er klar til brug.

8.3 Ikke-funktionelle krav

8.3.1 FURRPS+

Category	Details
Functionality	<ul style="list-style-type: none"> • Dynamisk justering af solpanelernes orientering ved hjælp af en toakset tracker (Use Case 1). • Realtidsovervågning og dataindsamling af miljøforhold som temperatur, luftfugtighed og lysintensitet. • Kompatibilitet med HTTP-servere og webbaserede dashboards.
Usability	<ul style="list-style-type: none"> • En brugervenlig webgrænseflade med realtidsgrafer og tabelvisning af data. • Intuitivt display på ESP32 (f.eks. temperatur og lysintensitet). • Tilgængelig via enhver webbrowser ved hjælp af ESP's IP-adresse.
Reliability	<ul style="list-style-type: none"> • Faldback-mekanisme til lokal datalagring på ESP32, hvis serveren er utilgængelig. • Diagnostiske tjek for aktuatorernes funktionalitet og sensorernes datakvalitet. • Kontinuerlig drift med automatiske justeringer og opdateringer på faste intervaller (f.eks. hver 10. sekund eller hvert minut). • Høj præcision i sensorernes dataindsamling (lysintensitet, temperatur og luftfugtighed). • Validering af data før behandling.
Performance	<ul style="list-style-type: none"> • Dataopdateringer på dashboardet med faste intervaller (f.eks. hvert minut). • Hurtig responstid til paneljusteringer baseret på sensorinput. • Optimeret brug af solpanelorientering for maksimal energiproduktion. • Effektivt strømforbrug for IoT-komponenterne. • Understøttelse af flere sensorer og mulighed for at udvide systemet til større opsætninger.
Supportability	<ul style="list-style-type: none"> • Modulært design med klar opdeling af funktionalitet (f.eks. sensorer, motorstyring og dataindsamling). • Nem udskiftning af komponenter (f.eks. Raspberry Pi, ESP32). • Mulighed for integration af ekstra sensorer eller komponenter i fremtiden (f.eks. vindmåler). • Understøttelse af tilføjelse af nye funktioner til dashboardet. • Klar dokumentation for systemopsætning, brug og fejlfinding.
+	<ul style="list-style-type: none"> • Juridiske krav: Systemet skal overholde GDPR for beskyttelse af brugernes data. • Testbarhed: Hvert krav skal være målbart og testbart, fx ved at verificere responstider og sensorvalidering. • Miljøhensyn: Materialer brugt i solpaneler og sensorer skal overholde standarder som RoHS. • Økonomisk overvejelse: Systemet skal være omkostningseffektivt uden at kompromittere kvalitet og pålidelighed.

Table 2: Requirements Overview for Solar Optimization System

8.4 Prioritering af funktionalitet(vha. MoSCoW)

8.4.1 MoSCoW

Kategori	Krav
Must have	<ul style="list-style-type: none">• Dynamisk justering af solpaneler for maksimal eksponering mod solen.• Realtidsovervågning af temperatur, luftfugtighed og lysintensitet.• Et webbaseret dashboard til visualisering af energiproduktion og miljødata.• Stabil og effektiv strømproduktion fra solpaneler.• Lokal datalagring som fallback, hvis serveren er utilgængelig (Use Case 2-udvidelse).• Automatisk recalibrering af panelpositioner hvert 10. sekund.
Should have	<ul style="list-style-type: none">• Sikker dataoverførsel mellem IoT-enheder og webklienten.• Brugergodkendelse for adgang til webklienten.• Integration med cloud-lagring for langvarig dataanalyse.• Advarsler på dashboardet for systemproblemer (f.eks. sensorfejl eller aktuatorfejl).
Could have	<ul style="list-style-type: none">• Mobilapp som supplement til det webbaserede dashboard.• Detaljerede energianalyser, såsom historiske data og trends.• Stemme- eller fjernkommandoer til manuelle systemjusteringer.• Integration med eksterne vejr-API'er til vejrsigter.• Datahistorik (mulighed for fremvisning af tidligere data).
Won't have	<ul style="list-style-type: none">• Integration med andre energikilder end solenergi (f.eks. vindmøller).• Offline-tilstand for fuld systemfunktionalitet.• Avancerede prædiktive algoritmer til energiproduktion baseret på vejrmonstre.• Integration med eksterne vejr-API'er til vejrsigter.

Table 3: Prioriterede krav til solcelleprojektet

9 Indledende analyser

9.1 Teknologianalyse

9.1.1 Hardware

Styreenhed - Raspberry

Et af vores projektkrav var brugen på et linux system, her har vi valgt at bruge raspberry pi zw. Da det er et godt valg til dataindsamling og aktuator kontrol, da den tilbyder indbygget Wi-Fi til real-time dataoverførsel, har mange GPIO pins til at styre både sensorer og aktuatorer til justering af solpaneler.

Desuden kører den på et Linux-operativsystem, hvilket gør det muligt med avanceret programmering og effektiv fejlhåndtering, samt har et relativt lavt strømforbrug, der gør den velegnet til

kontinuerlig drift i energibesparende systemer.

Fugtigheds / Temperatur-sensor

Da vi har lidt størrer udvalg, kan vi bruge en tabel form til at fordele de forskellige componenter:

**RH = står for Relativ Fugtighed. Det er et mål for, hvor meget fugt der er i luften i forhold til, hvor meget fugt luften maksimalt kan indeholde ved en given temperatur - skrevet i procent.*

Egenskab:	DHT11	DHT22	SI7021	HTU21
Temp interval	0 - 50 °C	-40 - 80 °C	-40 - 125 °C	-40 - 125 °C
Humidity interval	20 to 80% RH*	0 to 100% RH*	0 to 100% RH*	0 to 100% RH*
Temp nøjagtighed	±2 °C	±0.5 °C	±0.4 °C	±0.3 °C
Humidity nøjagtighed	±5% RH*	±2.5% RH*	±2% RH*	±2% RH*
Opløsning (hvor præcist den kan vise)	1 °C / 1% RH*	0.1 °C / 0.1% RH*	0.01 °C / 0.04% RH*	0.01 °C / 0.04% RH*
Output	Digital (single-wire)	Digital (single-wire)	I2C	I2C
Strøm Styring	3.3V - 5V	3.3V - 6V	1.9V - 3.6V	1.5V - 3.6V
Sampling rate (hvor mange gange per. Sek den opsamler data)	1 Hz	0.5 Hz	Up til 10 Hz	Up til 10 Hz
Kommentar	Begrænset i rækkevidde og nøjagtighed. Dog er det en billig løsning (ikke taget i betragtning)	Har bedre ydeevne til generelle applikationer	I7021 og HTU21 giver højst nøjagtighed og præcision, hvilket gør dem ideelle, men til en højere pris (ikke taget i betragtning)	

Lys sensor

Pga. der bruges en lidt størrer mængde af samme sensor, er der ikke så stort udvalg på skolens ressourcer. Systemet skal bruge 4 lys sensor for at kunne regulere sig iforhold til solens orientering. Her vælges PT550, som er en analog phototransistor sensor.

- God til at registrere lysændringer i simple systemer som at følge sollys til solceller. Hurtige reaktioner, som kan være godt til applikationer, hvor lysniveauet varierer. Hvilket er et nøgle punkt i vores projekt.
- Dog måler den ikke i Lux, hvilket er en standard lys enhed. Dette gør, at der selv skal kalibreres. Den har desuden ikke så god præcision som andre alternativer som BH1750 eller TSL2561

Egenskab	PT550
Type	Analog Sensor
Følsomhed	20 to 80% RH*
Output	Analog output, der varierer med lysintensiteten.
Strøm Styring	3.3V - 5V
Måleområde	1 Hz
Kommentar	Is cost-effective for simple applications but limited in range and accuracy.

Table 4: PT550 Sensor Specifikationer

Servo motor

Vi har valgt at bruge en servo motor, siden vi har brug for at solcellerne kan følge solen. Servo motoren består af et gearsystem og en feedback, der sikre præcis justering. Vi bruger servo motoren til at bevæge solpanelen horisontal.

Webserver

Til hosting af webserveren vælges en ESP32 microcontroller. Denne er kompakt, har lavt strømforbrug og er udstyret med indbygget WIFI, som muliggør hosting af en webserver på flere enheder samtidigt. Da der er større erfaring med ESP32 i forhold til Raspberry Pi (til web development), vælges ESP'en til projektet.

Esp32 er en velkendt, og modular form for microcontroller og der findes forskellige slags, i forhold til hvilke use-cases der står på spil.

Her har vi 4 til rådighed fra embedded stock, og vil beskrives i forhold til hvilke bedst passer til vores use-cases

1. *XIAO ESP32-C3*

- Lille formfaktor, som er Ideel til plads begrænsede opsætninger som IoT-sensor noder. Lavt strømforbrug, hvilket er velegnet til realtids dataindsamling og systemer der kræver lang batterilevetid.
- Begrænset kraft til mere komplekse opgaver som fejlbehandling eller UI

God til dataindsamling og sensor-drevne applikationer, men mindre egnet til realtidsjusteringer, der kræver mere processorkraft

2. *ESP32 LilyGO T-Display - endelige valg*

- Indbygget display, perfekt til visuel feedback af realtidsdata, som energiproduktion og systemstatus (use case 4). Dual-core processor: God til multitasking mellem dataindsamling og UI-visning.
- Begrænsede strømbesparende funktioner: Ikke så energieffektiv som XIAO ESP32-C3.

Stærk til UI-baserede applikationer, hvor visuel feedback af systemet er nødvendigt (hvilket var et could i vores MoSCoW). Mindre optimal til lavenergi sensor-drevne opgaver.

3. *ESP32-C6*

- Wi-Fi 6 support: Fremragende til stabil realtids dataoverførsel (use case 2) og fremtidssikret tilslutning. Thread-protokol: Understøtter mesh-netværk, hvilket kan være nyttigt til multisensorsystemer - dog har vi ikke stor brug for dette. Lavt strømforbrug: God til batteridrevne systemer, hvor kontinuerlig overvågning er nødvendigt.

God til dataoverførsel og fjernovervågning med fokus på realtids data indsamling og distribution over internettet.

4. *ESP32-S3 DevKitC*

- Dual-core processor med AI acceleration: Ideel til realtidsbehandling, såsom justering af solpanel orientering baseret på sensordata (use case 1). Mange GPIO pins vil passe til systemer, der kræver flere sensorer, aktuatorer og motorer (som justering af solpaneler). USB OTG, gør det let til fejlfinding og tilslutning af eksterne enheder eller sensorer
- Større størrelse gør det måske ikke egnet til små, plads begrænsede designs, og Højere strømforbrug mindre ideel til lavenergi applikationer, som langtids miljøovervågning.

Kan dog være overkill iforhold til hvad vi forventer systemet kan.

Her vælger vi at gå med LilyGO T-Display, da den har god visuel feedback (UI) og realtidsdata visning i form af display. Dette kan vi bruge til fejlfinding og vise vigtige oplysninger omkring vores system, som f.eks strømgenerering.

Strømsensor - INA169

Fordele:

1. Udfører strømmålinger på højside uden at påvirke ground.
2. Støtter et bredt spændingsområde (2,7 V - 60 V).
3. Enkel integration med minimalt behov for eksterne komponenter.
4. Energibesparende og pladsbesparende design.
5. Sikrer nøjagtige målinger med lineært analogt output og høj temperaturstabilitet.

Ulemper:

1. Afhængig af nøjagtigt dimensionerede eksterne modstande.
2. Mangler digitalt output, hvilket kræver brug af ADC.
3. Risiko for effekttab i shuntmodstanden ved høj strømstyrke.
4. Uegnet til systemer, der kræver elektrisk isolation.

INA169 er en pålidelig løsning til præcise og enkle strømmålinger i jordforbundne systemer, men der kan være behov for ekstra kredsløb til digitalisering eller elektrisk isolation.

Komponent	Beskrivelse
Styreenhed	Raspberry Pi ZW
Fugtigheds/Temp. sensor	HTU21
Servo	
Lyssensor	PT550
Strømsensor	INA169 breakout
Webserver	ESP32 LilyGO T-Display

Table 5: Komponent Liste

9.1.2 Software

VMware

VMware er en virtualiseringssoftware, der gør det muligt at oprette og køre flere virtuelle maskiner (VM'er) på en enkelt fysisk maskine. Dette giver udviklere mulighed for at simulere forskellige operativsystemer og miljøer uden at ændre noget på deres eksisterende opsætning. Det er særligt nyttigt i udviklingsprocessen, da man kan teste sin kode og drivere på forskellige platforme og konfigurationer, såsom forskellige versioner af Linux, uden at skulle geninstallere eller dual-boote. For din udvikling kan du kompilere og teste dine drivere i et kontrolleret, isoleret miljø, hvilket minimerer risikoen for at påvirke din primære maskine.

Visual Studio Code (VS Code)

VS Code er en alsidig, gratis teksteditor og IDE (Integrated Development Environment), der understøtter en bred vifte af programmeringssprog og teknologier, herunder HTML, CSS, JavaScript, og mange flere. Med sine mange udvidelser er VS Code yderst tilpasningsdygtig til forskellige udviklingsopgaver. Det bruges her til udvikling af både webserverkoden og Linux-systemet, og det understøtter nemt flere programmeringsparadigmer og frameworks. En populær udvidelse er Live Server Extension, der opretter en lokal webserver og viser ændringer i realtid, når du gemmer dit arbejde. Dette gør det let at teste UI-kode og lave hurtige ændringer uden at skulle genstarte eller opdatere manuelt.

PlatformIO/ExpressIDF

PlatformIO og ExpressIDF er udviklingsmiljøer, der er særligt rettet mod ESP32 microcontrolleren. De forenkler processen med at kompilere, uploade og administrere kode til embedded systemer som ESP32, hvilket gør dem ideelle til IoT-projekter. Disse værktøjer hjælper med at håndtere biblioteker og frameworks og gør det muligt at kompilere koden og uploade den til ESP32 hurtigt og nemt. Dette betyder, at udviklere ikke skal bekymre sig om den komplekse opsætning af miljøet hver gang, men kan fokusere på kodens funktionalitet.

HTML (HyperText Markup Language)

HTML er et markup-sprog, der bruges til at skabe strukturen i et webinterface. Det definerer, hvordan elementer som overskrifter, afsnit, tabeller og billeder præsenteres på websiden. I dit tilfælde bruges det til at bygge brugergrænsefladen, hvor brugeren kan interagere med systemet, f.eks. til at vise informationer om solpanelernes status eller vejrdata fra sensorer.

Cascading Style Sheets

CSS bruges til at style HTML-elementerne og give webinterfacet et pænt layout. Det kan justere alt fra skrifttyper og farver til komplekse layoutændringer og animationer. CSS hjælper med at gøre interfacet visuelt tiltalende og funktionelt på tværs af forskellige skærmstørrelser og enheder.

JavaScript

JavaScript bruges til at tilføje interaktivitet til webinterfacet. Det kan f.eks. håndtere brugerhandling som klik på knapper, indtastning af data, eller dynamisk ændring af indhold uden at genindlæse hele siden. I dette projekt kan JavaScript bruges til at opdatere solpanelernes status i realtid eller interagere med sensorer for at vise aktuelle vejrforhold.

Live Server Extension

Live Server Extension er en udvidelse til VS Code, der opretter en lokal HTTP-server til at vise ens webside i realtid. Når man gemmer ændringer i HTML, CSS eller JavaScript-kode, opdateres websiden automatisk i browseren. Dette gør det nemt at teste ændringer med det samme uden at skulle genindlæse eller opdatere manuelt, hvilket forbedrer udviklingsoplevelsen og sparer tid.

9.2 Risikoanalyse

9.2.1 Projekt omfang

Vi har fokuseret på at lave et projekt, der primært passer til vores samlede faglige kompetencer. Dette er afgørende for, at vi kan færdiggøre projektet inden for den bestemte tidsramme og med den nødvendige kvalitet. Samtidig har vi ikke været bange for at udfordre os selv ved at inkludere enkelte elementer, der ligger uden for den viden og de færdigheder, vi har opnået gennem vores studieforløb, så længe disse elementer ikke gør projektet unødigt komplekst. Vi har desuden samtaler med vejlederen løbende gennem projektforløbet og modtager værdifuld feedback, som vil indgå i vores vurdering af, om projektets omfang og sværhedsgrad er passende.

9.2.2 Tidspres

Under dette arbejdsforløb kan der opstå pres, da der er mange afleveringer i dette semester, hvis tiden ikke disponeres korrekt. Derfor er det vigtigt at have en tidsplan, som kan følges. Dette er med til mindske risikoen for at ryge bagud med projektet.

9.2.3 Management

Vi har udpeget en Scrum Master og en projektleder for at sikre struktur, fremdrift og effektivitet i arbejdet. Scrum Masteren fokuserer på at facilitere teamets arbejdsprocesser, mens projektlederen sikrer, at opgaver prioriteres korrekt, og tidsplanen overholdes. Dette reducerer risikoen for forvirring og ineffektivitet.

9.2.4 Teamwork/Samarbejde

Uenigheder i teamet eller ineffektivt samarbejde kan opstå og vil have en negativ indvirkning på projektets fremdrift, kvalitet og arbejdsdynamik. For at minimere denne risiko har vi etableret klare kommunikationslinjer. Der bliver derfor særdeles lagt vægt på konfliktløsninger når/hvis de opstår.

9.2.5 Frafald eller fravær

For at sikre kontinuitet og minimere risikoen for forsinkelser ved fravær eller frafald, har vi valgt at tilknytte to personer til hver del af projektet. Dette bidrager til, at projektet kan fortsætte uden større afbrydelser og sikrer, at arbejdet forløber så gnidningsfrit som muligt.

10 Systemarkitektur

10.1 Domænemodel

For at udarbejde domænemodellen i figur 7 er der først gennemført en system-domæneanalyse. Formålet med denne analyse er at identificere og afgrænse de vigtigste begreber og relationer i Smart Wagon-systemet. Domænemodellen illustrerer relationerne mellem IoT-DASTS og dets omgivelser samt repræsenterer de centrale begreber i systemet.

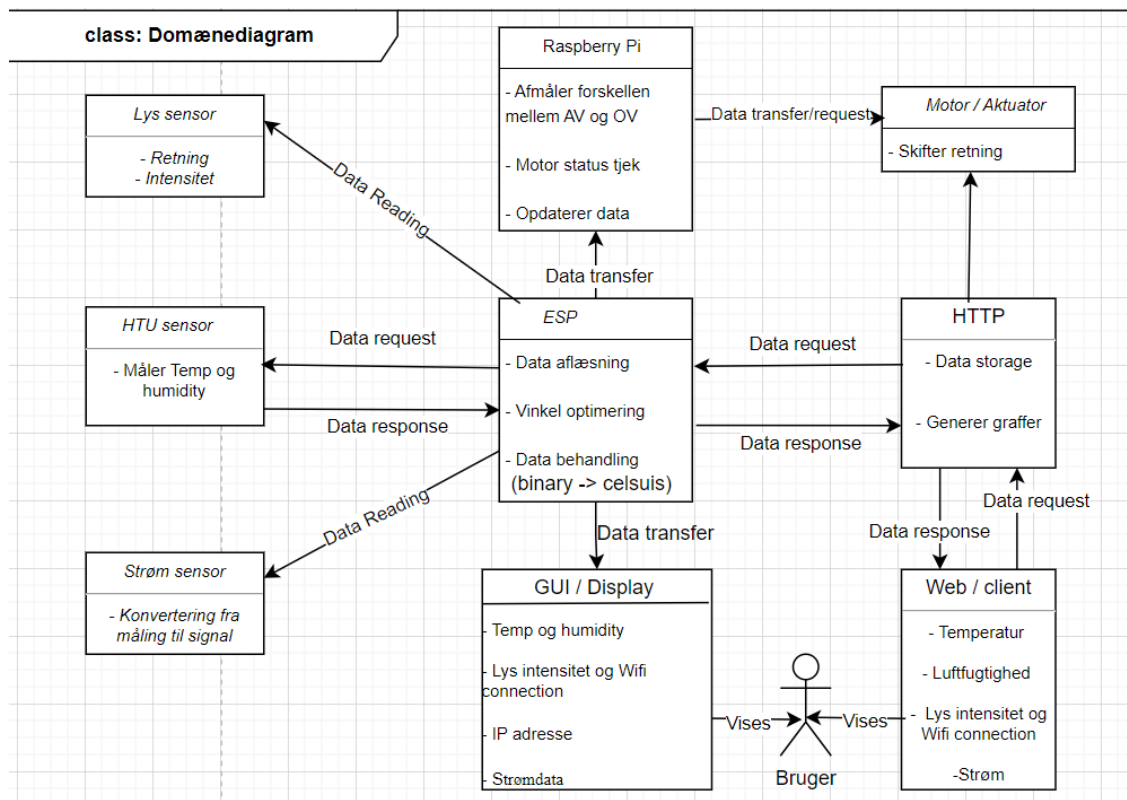


Figure 7: Domænemodel

10.2 BDD'er og IBD

10.2.1 Block Definition Diagram

BDD'et beskriver systemets komponenter og deres interaktioner for at vise, hvordan data flyder gennem systemet, og hvordan forskellige moduler arbejder sammen for at opnå optimal solsporing og energiproduktion.

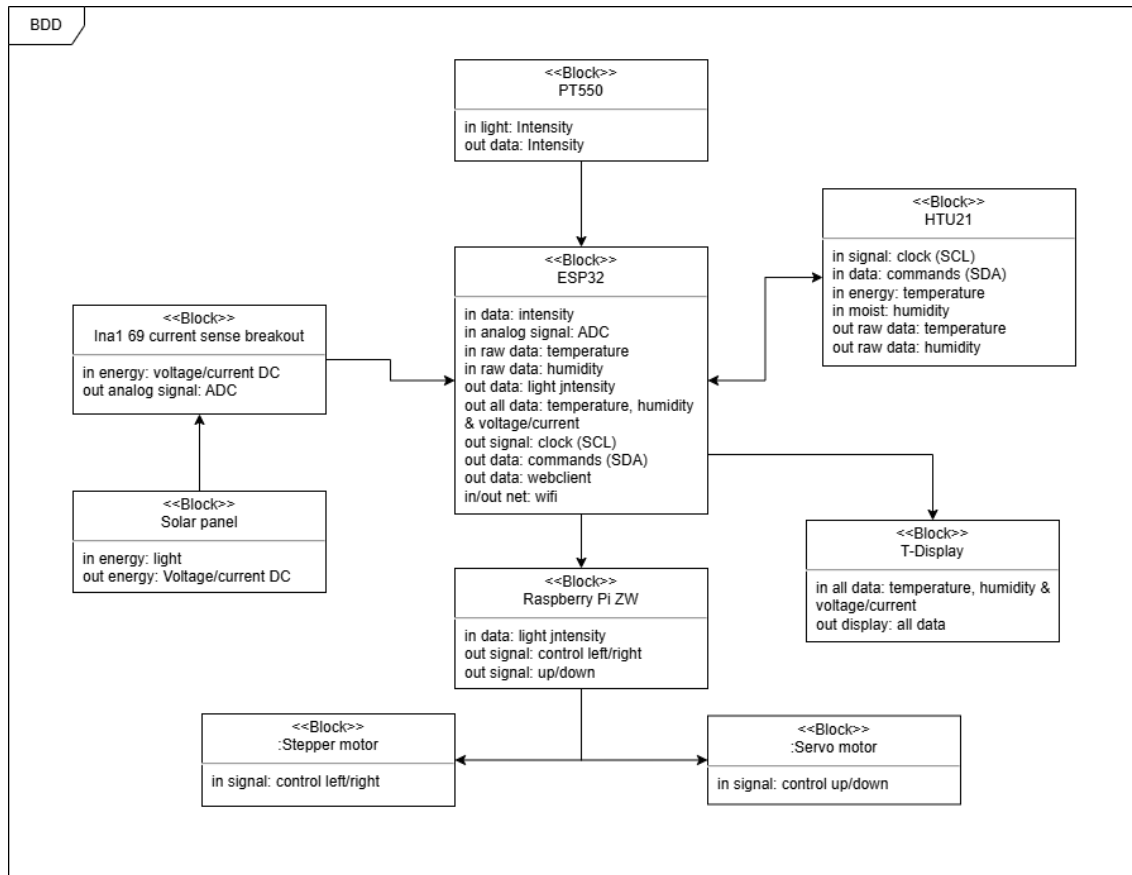


Figure 8: Block Definition Diagram

10.2.2 Internal Block Diagram

IBD'et illustrerer, hvordan systemet kombinerer miljødata, energidata og aktuator kontrol for at optimere solcellens effektivitet gennem dynamisk orientering og brugervenlig overvågning. Hver komponent spiller en rolle i at sikre, at solcellen altid er korrekt justeret for maksimal energiproduktion.

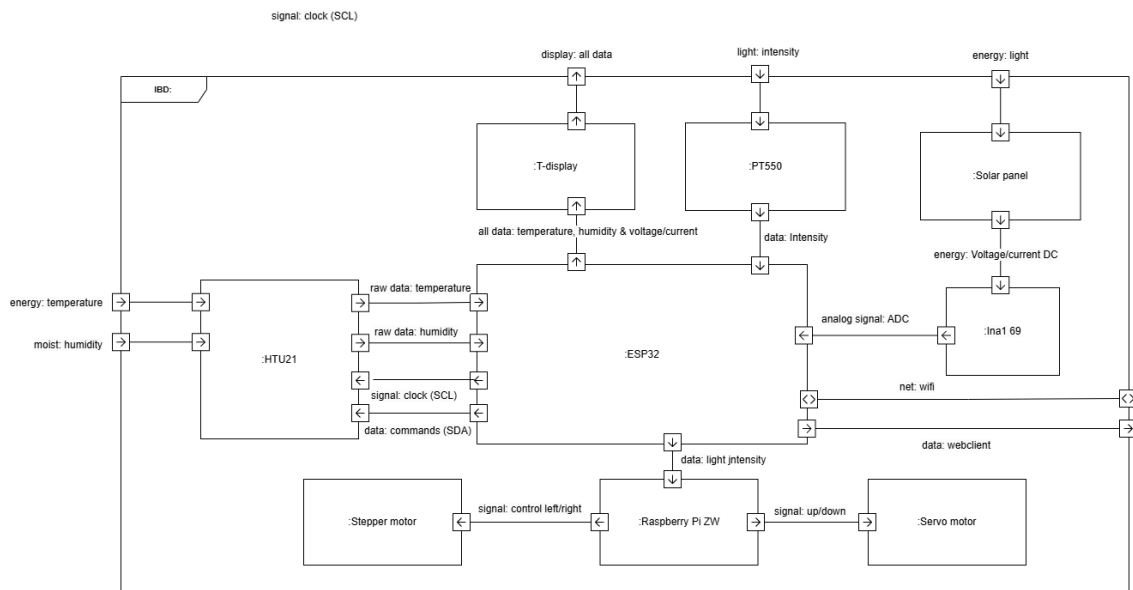


Figure 9: beta IBD

10.3 Sekvensdiagrammer for handling

Fire sekvensdiagrammer er blevet udarbejdet for at beskrive IoT-DASTS' adfærd gennem handlinger baseret på de fire Use Case-diagrammer præsenteret i afsnit 8, Kravspecifikation. Da rapportens fokus primært har været på UC1 og UC2, vil kun sekvensdiagrammerne for disse blive præsenteret i dette afsnit. De øvrige to sekvensdiagrammer kan findes i bilagsrapporten under afsnittet om systemarkitektur.

10.3.1 Sekvensdiagram for UC1

Sekvensdiagrammet for use case 1 viser, hvordan vores system arbejder iterativt i et 100 ms-loop for at justere solpanelernes position baseret på den aktuelle lysintensitet. Det illustrerer, hvordan lyssensorer, ESP, Raspberry Pi, og aktuatorer samarbejder for at sikre, at solpanelerne er korrekt orienteret.

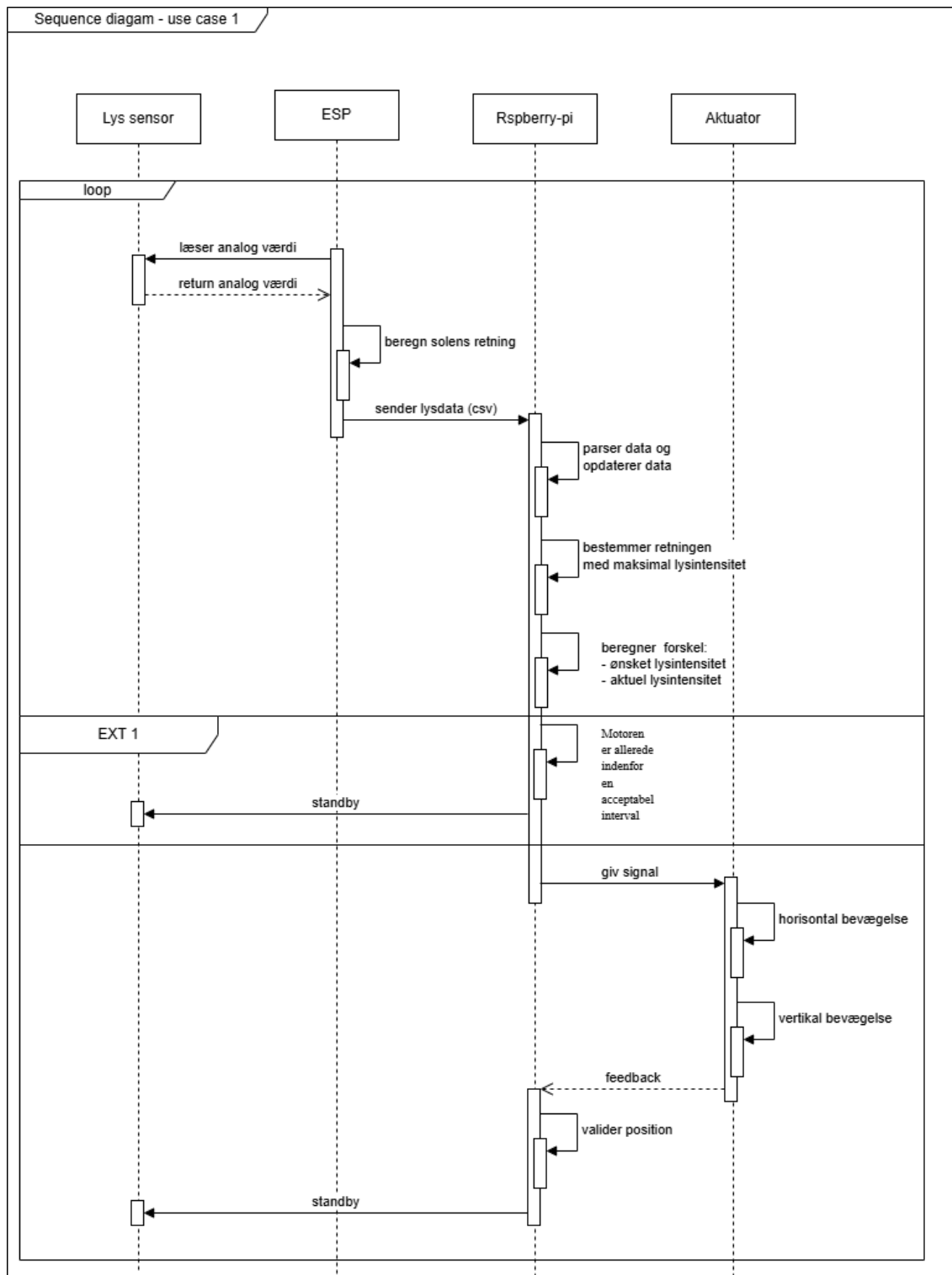


Figure 10: beta sd1

10.3.2 Sekvensdiagram for UC2

Dette sekvensdiagram for use case 2 fremhæver, hvordan systemet sikrer korrekt og kontinuerlig indsamling, validering og distribution af miljødata. Interaktionen mellem sensorer, ESP og HTTP-server understøtter robust og pålidelig databehandling, hvilket er afgørende for systemets funktionalitet.

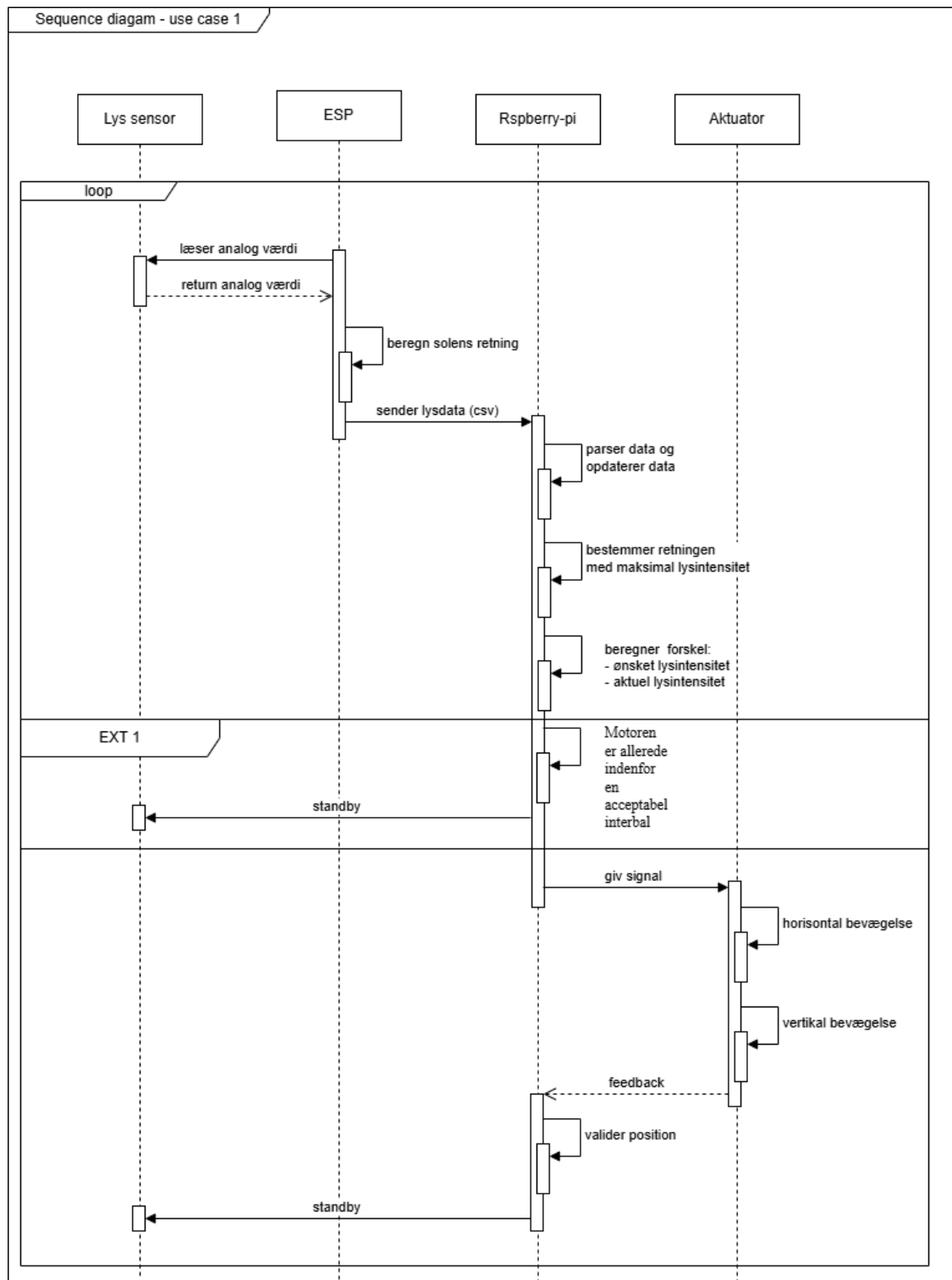


Figure 11: beta SD2

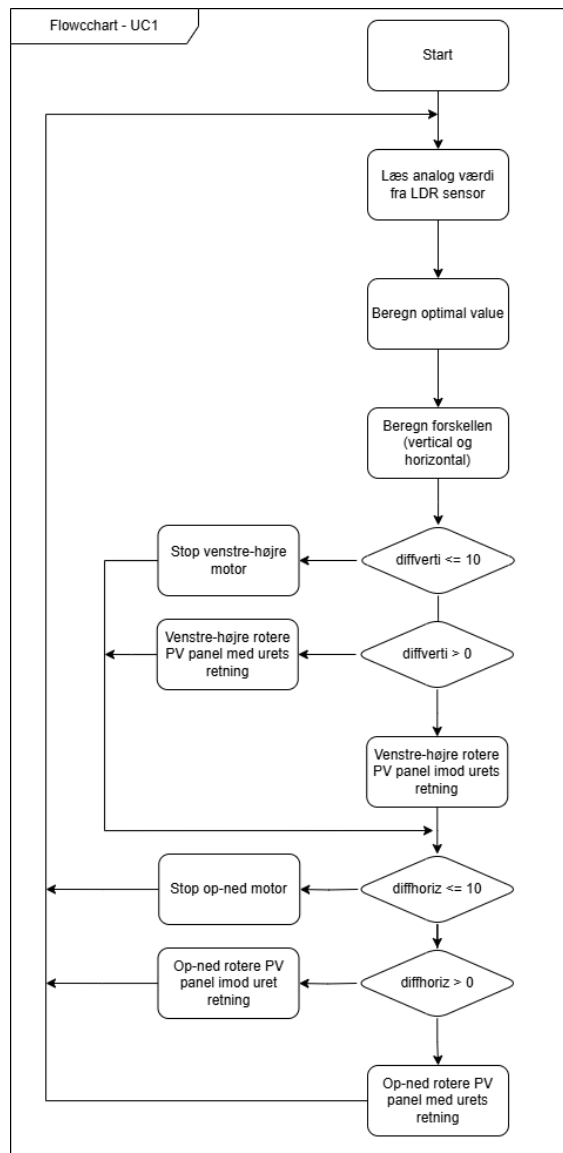


Figure 13: Flowchart - bevægelse

11.3.2 Applikationsmodel baseret på UC1 og UC2

I dette afsnit præsenteres den udarbejdede applikationsmodel for IoT-DASTS. Præsentationen omfatter et klassediagram samt et sekvensdiagram med metoder for UC1 og UC2. Applikationsmodellerne for de øvrige Use Cases findes i bilagsrapportens afsnit Revurdering af applikationsmodellen for IoT-DASTS.

UML klassediagram for UC1

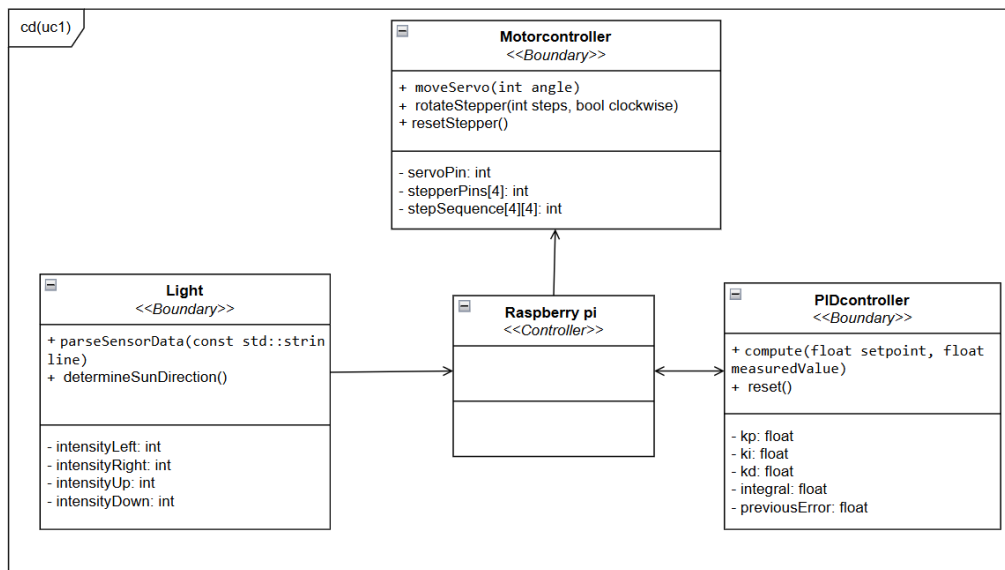


Figure 14: Klassediagram for Uc1

Sekvensdiagram med metoder for UC1

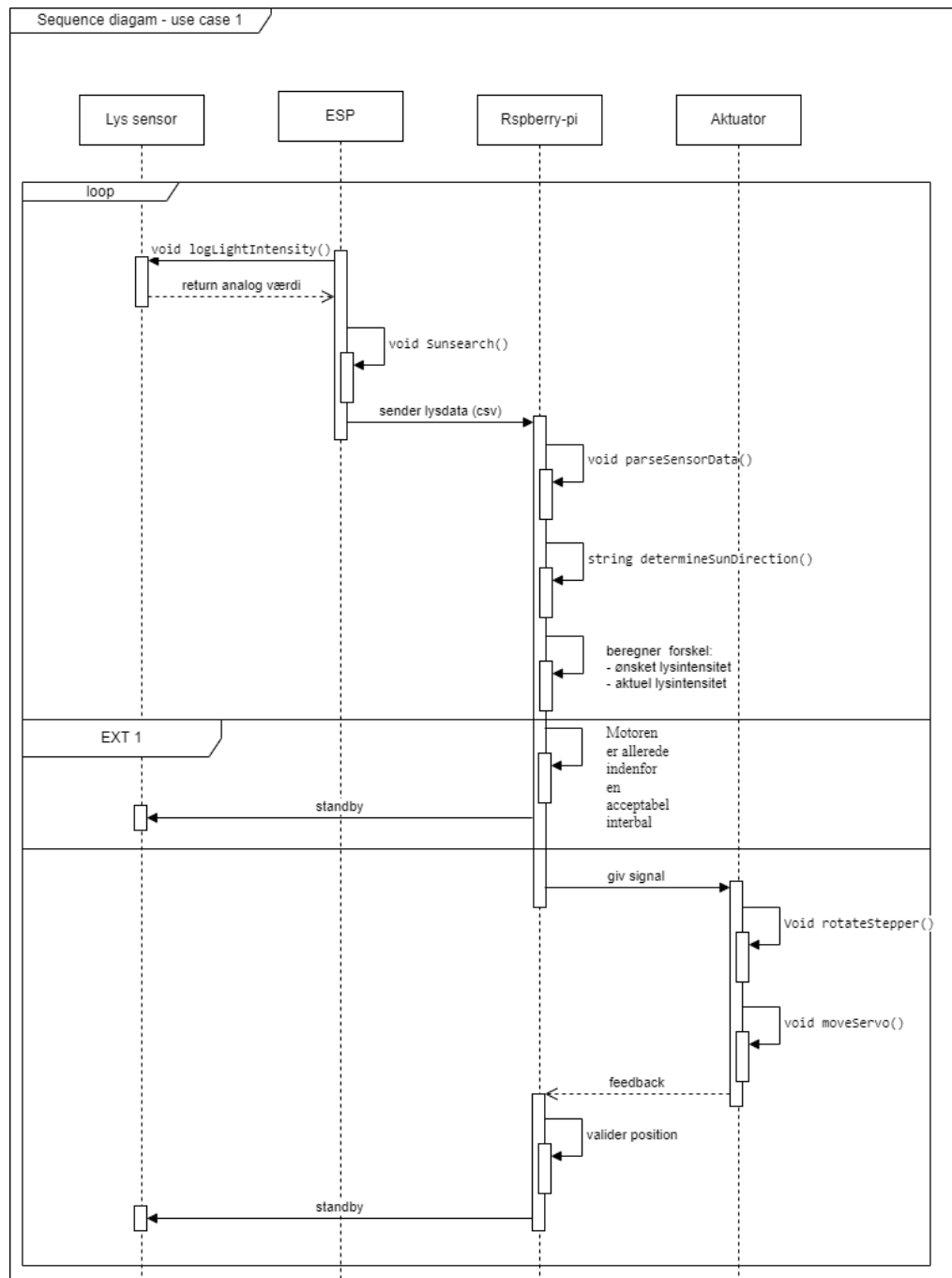


Figure 15: Klasse sekvens diagram - bevægelse

UML klassediagram for UC2

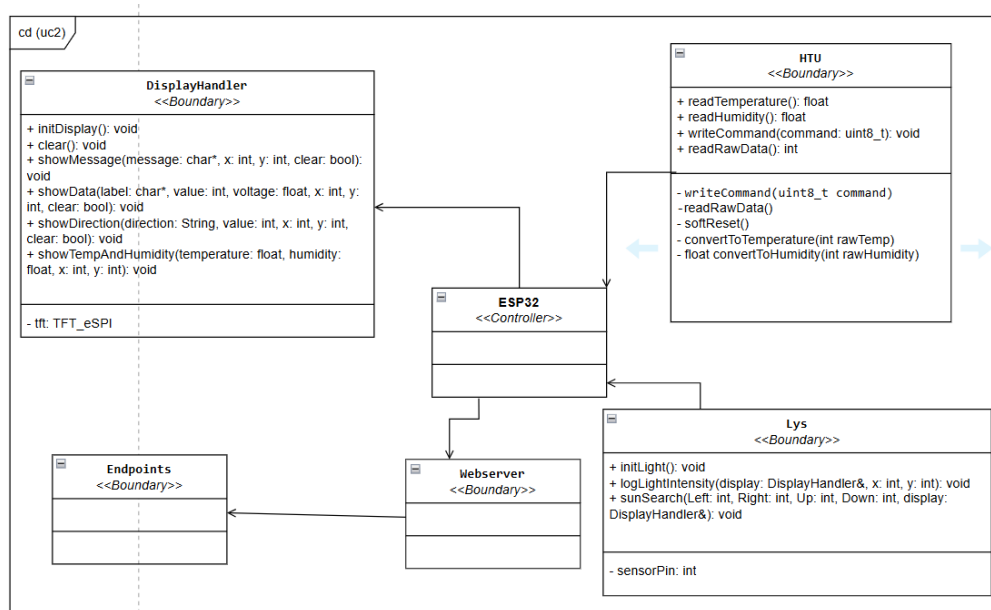


Figure 16: Klassediagram for UC2

Sekvensdiagram med metoder for UC2

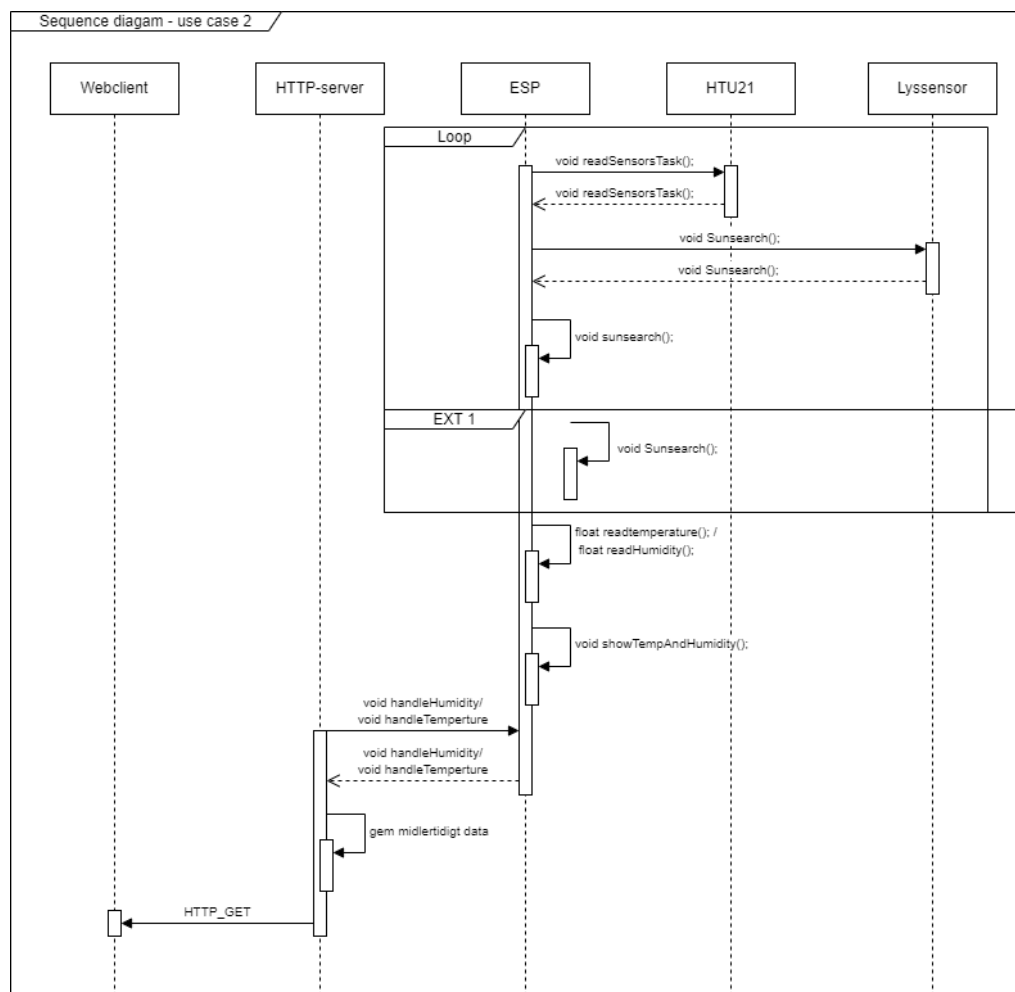


Figure 17: Klasse sekvens diagram - dataindsamling

12 Realisering af systemet

12.1 Hardware implementering

Implementering af motor (stepper og servo): Motorerne blev implementeret for at styre mekaniske bevægelser i projektet. Stepper-motoren blev valgt for præcis positionskontrol, mens servoen blev brugt til hurtige og præcise vinkeljusteringer. Begge motorer blev drevet via en motor-driver og styret med PWM-signaler fra ESP'en. UART-forbindelsen blev brugt til at sende kommandoer til motorerne, men platform-driverens opsætning krævede yderligere fejlfinding.

Implementering af HTU21: HTU21-sensoren blev implementeret for at måle temperatur og luftfugtighed. Sensoren kommunikerer via I2C, hvilket gjorde integrationen med ESP'en nem og effektiv. Dataen fra sensoren blev periodisk læst og sendt til displayet for visualisering samt til UART-forbindelsen for videre brug i systemet.

Implementering af lyssensor (PT550): Lyssensoren PT550 blev brugt til at måle lysintensitet. Sensorens analoge output blev forbundet til en ADC-pin på ESP'en, som konverterede signalet til en digital værdi. Denne værdi blev analyseret for at justere solpanelernes retning, hvilket forbedrede energieffektiviteten i systemet.

Implementering af ESP: ESP'en fungerede som den centrale styringsenhed i systemet. Den håndterede kommunikation med de forskellige sensorer og aktuatorer, bearbejdede data og styrede output baseret på inputs fra UART og sensorer. ESP'en blev også brugt til at vise realtidsdata på displayet og til at sende diagnosticeringsinformation under testfasen.

12.2 Software implementering

I dette afsnit beskrives softwareimplementeringen af IoT-DASTS' delsystemer. Den tilhørende kode er vedlagt i bilaget.

12.2.1 Implementering af webclienten

```
void handleTemperature(AsyncWebServerRequest *request) {  
    float temp = sensor.readTemperature();  
    if (isnan(temp)) {  
        request->send(500, "text/plain", "Failed to read temperature");  
    } else {  
        request->send(200, "text/plain", String(temp));  
    }  
}
```

Figure 18: Eksempel funktion: handleTemperature

12.2.2 1. Server-Side: Hndter Temperaturforesprgsler

- Funktionen handleTemperature kobles til endpointet temperature med server.on.
- Når der sendes en GET-forespørgsel til temperature, læser funktionen temperaturen via sensor.readTemperature().
- Hvis aflsningen fejler, sendes en 500-fejl som svar. Ellers sendes temperaturværdien som ren tekst.

```
server.on("/", HTTP_GET, handleRoot);  
server.on("/temperature", HTTP_GET, handleTemperature);  
server.on("/graph_Humidity", HTTP_GET, handleHumidity);  
server.on("/humidity", HTTP_GET, handleHumidity);  
server.on("/graph_Temp", HTTP_POST, handleHumidity);
```

Figure 19: Billede af server init

For at init vores server bruger vi server.on metoden.

12.2.3 Implementering af motor

servo-stepper.c er en platformdriver designet til at styre både en servomotor og en steppermotor ved hjælp af GPIO-pins, med karakterenheder oprettet for hver motor. Den metode vi har brugt, til styring af solcellerne er ved hjælp af en servo/stepper combination. der gør det muligt at bevæge solcellerne i både x og y retning.

Til at gøre dette bruger vi en Raspberry Pi med et linux system, og laver nogle drivere vi bl.a. har brug faget HAL, Heraf gør vi brug af read/write file operations men også det at kunne probe vores modul in på raspberry pi. her opdeler vi koden i Kernal space og Kernal space kode:

1. GPIO Write `gpio_write` er en funktion, der håndterer skriveoperationer fra User-space til motorerne via `/dev`-filer. Denne funktion analyserer data fra brugeren og udfører den nødvendige motorstyring.

For Servomotoren:

- Modtager en vinkel (0-180 grader) som input.
- Beregner den tilsvarende PWM-duty-cycle baseret på vinklen.
- Aktiverer og deaktiverer GPIO-pins (duty-cycle) for at generere det korrekte PWM-signal.

For Steppermotoren:

- Modtager en kommando som "`forward 50`" (retning og antal trin).
- Parser input og vælger retning.
- Aktiverer GPIO-pins sekventielt baseret på en trinsekvens for at rotere motoren.

2. Probe Funktion `plat_drv_probe` initialiserer driveren og forbinder den til hardwaren, når enheder tilføjes (f.eks. under opstart).

For Servomotoren:

- Henter GPIO-pinsen fra device tree via `of_get_named_gpio`.
- Anmoder om kontrol over servoens GPIO-pin og opsætter den som output.
- Tildeler en `/dev`-fil (`/dev/plat_drv0`) til brug for User-space.

For Steppermotoren:

- Henter en base-GPIO fra device tree for steppermotorens pins.
- Anmoder om kontrol over GPIO-pins for alle fire faser af motoren.
- Opsætter `/dev`-filer (`/dev/plat_drv1` til `/dev/plat_drv4`) for steppermotorens kontrol.

3. Remove Funktion `plat_drv_remove` rydder op, når driveren fjernes, og sikrer, at alle ressourcer frigøres.

For Servomotoren: Frigiver den GPIO-pin, der blev brugt til servomotoren.

For Steppermotoren:

- Frigiver alle fire GPIO-pins, der blev brugt til steppermotoren.
- Fjerner tilknyttede `/dev`-filer for at forhindre yderligere adgang.

User-Space Driveren binder sig til de relevante GPIO-pins under initiering, opsætter karakterenheder for hver motor, og sikrer, at hardwareinteraktioner udføres effektivt og stabilt, samtidig med at den muliggør kommunikation mellem kernel- og User-space.

For **servomotoren** håndterer driveren positionskontrol ved at modtage en vinkel (0-180 grader) fra User-space via `/dev/plat_drv0`. Den beregner herefter en tilsvarende PWM-duty-cycle og genererer et signal, der justerer motoren præcist til den ønskede vinkel. PWM-signalet er baseret på perioder, der tilpasses dynamisk, hvilket sikrer nøjagtighed.

For **steppermotoren** implementerer driveren en trinsekvens, der aktiverer GPIO-pins i rækkefølge for at rotere motoren med et specifikt antal trin i enten fremadgående eller baglæns retning. Brugeren sender kommandoer som "`forward 50`" eller "`backward 50`" via `/dev/plat_drv1` til `/dev/plat_drv4`. Driveren udfører disse kommandoer ved at skifte GPIO-pins baseret på en predefined sekvens, mens den respekterer timingkravene for motoren.

```

float readTemperature() {
    if (sensorFound) {
        return htu21d.getTemperature(); // Returns temperature in Celsius
    } else {
        Serial.println("Error: HTU21D sensor not found.");
        return NAN;
    }
}

```

Figure 20: Enter Caption

```

// Read humidity as a percentage
float readHumidity() {
    if (sensorFound) {
        return htu21d.getHumidity(); // Returns humidity as percentage
    } else {
        Serial.println("Error: HTU21D sensor not found.");
        return NAN;
    }
}
};

```

Figure 21: Enter Caption

12.2.4 Implementering af HTU21

HTU Koden indeholder to væsentlige funktioner, der læser temperatur og fugtighed fra en HTU21D-sensor:

`readTemperature()` |:

- Returnerer temperaturen i Celsius, hvis sensoren er fundet (`sensorFound` er sand).
- Hvis sensoren ikke findes, printes en fejlmeddelelse til Serial, og `NAN` returneres.

`readHumidity()` |:

- Returnerer luftfugtigheden som en procentdel, hvis sensoren er fundet.
- Ved fejl printes en meddelelse, og `NAN` returneres.

Disse funktioner bliver b.l.a. brugt til at sende data til webclient, display og serial debugging gennem terminal.

12.2.5 Implementering af lyssensor

Lyssensorerne udgør et af de vigtigste komponenter i IoT-baserede Dual-Axis Solar Tracking System (IoT-DASTS). De er ansvarlige for at måle lysintensiteten i fire retninger – venstre, højre, op og ned – og leverer data til systemet, der bruges til at justere solpanelernes orientering dynamisk. Dette er centralt for at opnå maksimal energiproduktion.

Lyssensorerne sikrer, at systemet løbende registrerer solens position og foretager realtidsjusteringer af solpanelerne. Dette sker ved at analysere sensorernes data og identificere den retning med den højeste lysintensitet. PT550-sensorerne, som bruges i projektet, er valgt på grund af deres

lave pris og hurtige reaktion på ændringer i lysniveauet, hvilket gør dem velegnede til applikationer som denne.

Lyssensorens implementering er opdelt i tre hoveddele: Der er initialisering, databehandling og retning og styring.

For at sikre nøjagtige målinger er ADC på ESP32 konfigureret med en opløsning på 12 bit og en attenuering på 11 dB. Dette giver præcise aflæsninger og understøtter måling af spændinger op til 3,3 V.

```
// Constructor to initialize the pin
LightSensor(int pin) : sensorPin(pin) {}

void initLight() {
    adc1_config_width(ADC_WIDTH_BIT_12);
    // Set 11 dB attenuation
    adc1_config_channel_atten(ADC1_CHANNEL_0, ADC_ATTEN_DB_12); // GPIO36
    adc1_config_channel_atten(ADC1_CHANNEL_1, ADC_ATTEN_DB_12); // GPIO35
    adc1_config_channel_atten(ADC1_CHANNEL_4, ADC_ATTEN_DB_12); // GPIO32
    adc1_config_channel_atten(ADC1_CHANNEL_5, ADC_ATTEN_DB_12); // GPIO33
    Serial.println("ADC1 channels configured with 11 dB attenuation");
}
```

Figure 22: Snippet 1

I setup()-funktionen initialiseres hver af de fire sensorer.

```
leftSensor.initLight();
rightSensor.initLight();
upSensor.initLight();
downSensor.initLight();
```

Figure 23: Snippet 2

Denne konfiguration sikrer, at sensorerne er klar til at levere stabile og præcise data til systemet.

I koden aflæses data fra de fire sensorer. Dataene behandles ved at konvertere den rå ADC-værdi til en tilsvarende spænding. Disse resultater logges på den serielle monitor og vises på systemets TFT-skærm. Den implementerede kode ses her:

```

// Method to read and log the light intensity, also display on TFT
void logLightIntensity(DisplayHandler& display, int x, int y) {
    int sensorValue = analogRead(sensorPin);
    float voltage = sensorValue * (3.3 / 4095.0);

    // Display data on TFT screen
    display.showData("Light Intensity", sensorValue, voltage, x, y);

    // Example logic based on sensor value (for serial monitor)
    if (sensorValue > 3000) {
        Serial.println("High light intensity - solar panels adjusted optimally.");
        Serial.println(sensorValue);
    } else if (sensorValue < 1000) {
        Serial.println("Low light intensity - consider changing solar panel direction.");
        Serial.println(sensorValue);
    }
}

```

Figure 24: Snippet 3

I loop()-funktionen kaldes logningsfunktionen for hver sensor:

```

// Log light intensities
leftSensor.logLightIntensity(display, 0, 30);
rightSensor.logLightIntensity(display, 0, 40);
upSensor.logLightIntensity(display, 0, 50);
downSensor.logLightIntensity(display, 0, 60);

```

Figure 25: Snippet 4

Disse målinger giver et realtidsbillede af lysintensiteten, som er essentiel for systemets styring. Når data fra sensorerne er logget, anvendes de til at bestemme retningen med den højeste lysintensitet. Dette udføres af Sunsearch-funktionen, som sammenligner sensorværdierne og identificerer den optimale retning. Resultatet vises på TFT-skærmen og sendes til motorstyringen, som justerer panelerne.

```

// Find the sensor with the highest intensity and display it
void Sunsearch(int Left, int Right, int Up, int Down, DisplayHandler& display) {
    int maxIntensity = Left;
    String direction = "Venstre"; // Left

    if (Right > maxIntensity) {
        maxIntensity = Right;
        direction = "Højre"; // Right
    }
    if (Up > maxIntensity) {
        maxIntensity = Up;
        direction = "Op"; // Up
    }
    if (Down > maxIntensity) {
        maxIntensity = Down;
        direction = "Ned"; // Down
    }

    // Output the result on the TFT display
    display.showDirection(direction, maxIntensity, 10, 100);

    // You can also print this to the serial monitor if needed
    Serial.print("Maximum intensity is in direction: ");
    Serial.print(direction);
    Serial.print(" with value: ");
    Serial.println(maxIntensity);
}
};

```

Figure 26: Snippet 5

Funktionen integreres i hovedsløjfen og sikrer, at panelerne altid følger den mest lysintensive retning, og på denne måde kan problemstillingen opnås ved at maksimere energiproduktionen ved hjælp af dynamisk panelstyring.

13 Tests og resultater

13.1 Modultest

For at teste, om vores ESP-moduler fungerer korrekt, har vi individuelt tilsluttet og testet hver modul med ESP'en. Dette inkluderer lyssensoren, HTU-sensoren og displayet.

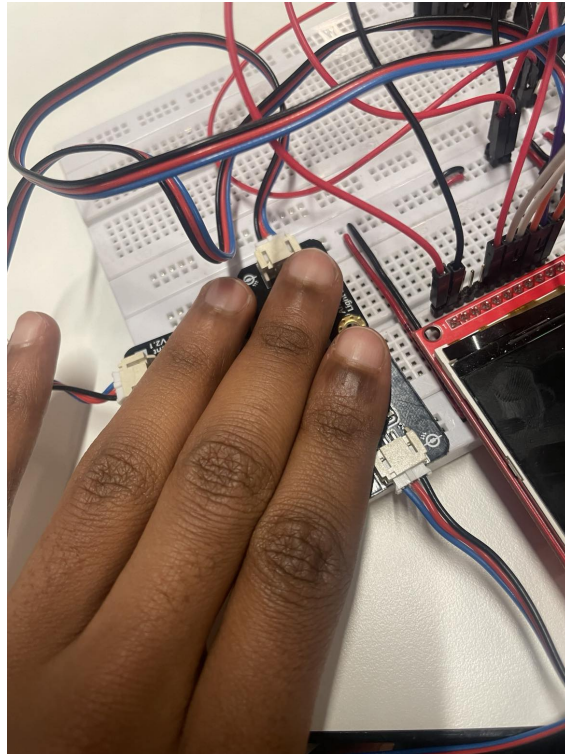


Figure 27: Enter Caption

```
Low light intensity - consider changing solar panel direction.  
-----  
Low light intensity - consider changing solar panel direction.  
-----  
Low light intensity - consider changing solar panel direction.  
-----  
Low light intensity - consider changing solar panel direction.  
-----  
Maximum intensity is in direction: Venstre with value: 373
```

Figure 28: Test af lys sensor, Low light condition

Vi har via Serial Monitor i PlatformIO testet kommunikationen. Ved at placere hånden over lyssensorerne får vi som resultat et lavt *light intensity response* fra ESP'en.

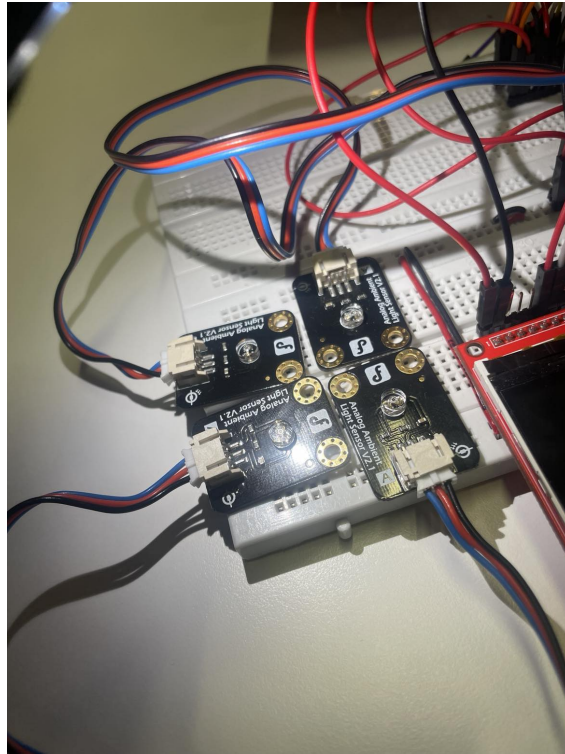


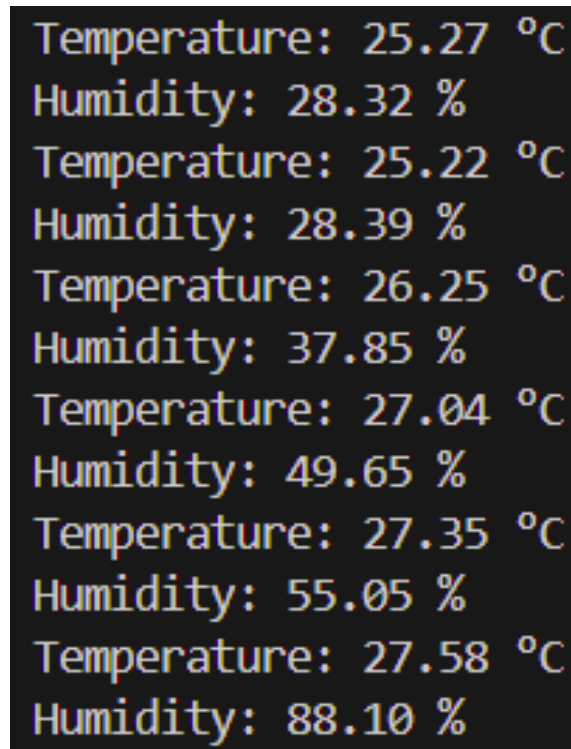
Figure 29: Enter Caption

Vi kan desuden identificere, hvilken lyssensor der modtager mindre lys end de andre.

```
High light intensity - solar panels adjusted optimally.
4095
High light intensity - solar panels adjusted optimally.
4095
High light intensity - solar panels adjusted optimally.
4095
High light intensity - solar panels adjusted optimally.
4095
Maximum intensity is in direction: Venstre with value: 4095
```

Figure 30: Test af lys sensor, High light condition

Og i dette tilfælde har vi fjernet hånden fra sensorerne og rettet en telefonlommelygte mod dem. Dette resulterede i en maksimal aflæsning for alle sensorerne.



Temperature (°C)	Humidity (%)
25.27	28.32
25.22	28.39
26.25	37.85
27.04	49.65
27.35	55.05
27.58	88.10

Figure 31: Test af HTU sensor (stigende)

HTU-sensoren blev testet ved at placere en finger på sensoren, hvilket resulterede i en lille temperaturstigning på ca. 2 grader.

For den sidste *humidity reading* åndede vi på sensoren, hvilket medførte en stigning i luftfugtighedsniveauet.

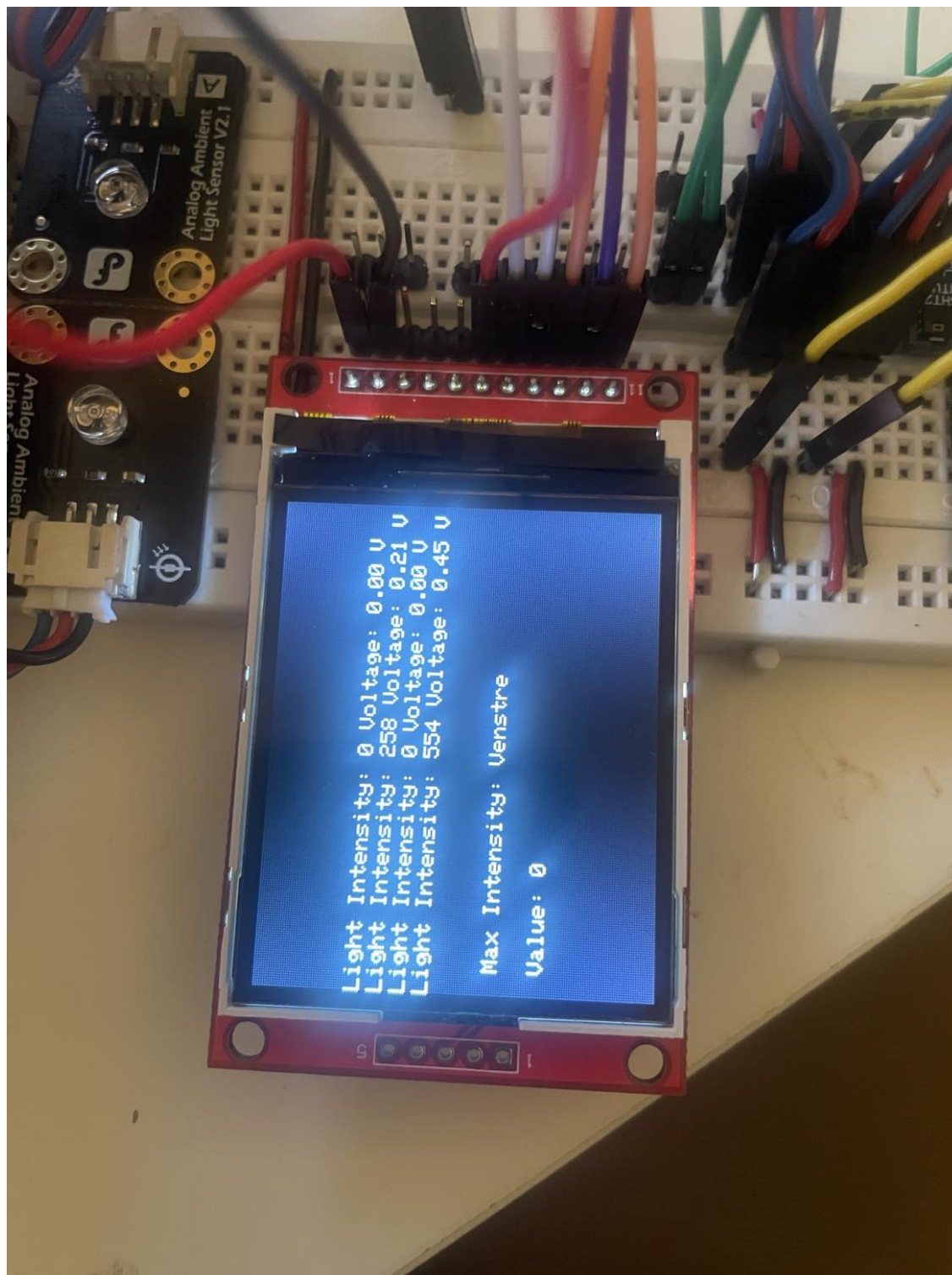


Figure 32: Test af display + sensor data

Vi testede displayet ved at undersøge, om vi kunne vise dataen fra HTU-sensoren, hvilket fungerede som planlagt.

I projektet stødte vi på en udfordring med at få motoren til at bevæge sig baseret på data

modtaget via UART. Problemet skyldtes, at vi ikke kunne initialisere vores platform-driver korrekt til at åbne og læse GPIO-data gennem UART.

13.2 Accepttest

Dette afsnit præsenterer den udarbejdede accepttest for IoT-DASTS samt en konklusion på resultaterne af den udførte test. Accepttesten fokuserer på systemets funktionalitet.

13.2.1 Accepttest for UC1 - Bevægelse

Test-NR.	Testbeskrivelse	Forventet resultat	Faktisk observation	Vurdering
UC1-T1	Lyssensor måler lysintensitet .	Sensor returnerer valide data.	Lyssensorer registrerer lysintensitet i fire retninger: Venstre, Højre, Op, Ned.	OK
UC1-T2	ESP validerer målinger og beregner optimal vinkel.	ESP sender korrekt vinkel til Raspberry Pi.	Kommandoer bliver sendt til Raspberry Pi, f.eks. omdrejning til venstre/højre eller op/ned	OK
UC1-T3	Raspberry Pi styrer aktuatorerne til optimal position.	Solpanel justeres korrekt, og positionen valideres.	Ikke opfyldt	Delvis
UC1-T4	Raspberry Pi detekterer, at den optimale vinkel allerede er opnået.	Ingen aktuatorbevægelse finder sted, og systemet går i standby.	Det virker, men den drejes ikke til den nøjagtige position	NEJ
UC1-T5	Lyssensor leverer ugyldige data (f.eks. mørke eller obstruktion).	Systemet bruger seneste kendte gyldige data og logger fejlen.	Ja, det virket som forventet.	Ja
UC1-T6	Aktuatoren fejler under justering.	Systemet logger fejlen, afbryder operationen og forsøger senere.	Ikke opfyldt	NEJ
UC1-T7	Raspberry Pi sender justeringskommandoer i mørke (natten).	Systemet udfører ingen justeringer og logger inaktivitet.	Ikke opfyldt	NEJ

13.2.2 Accepttest for UC2 - Strømgenerering

Test-NR.	Testbeskrivelse	Forventet resultat	Faktisk observation	Vurdering
UC2-T1	Temperatur- og luftfugtighedssensor leverer data til ESP.	Data indsamles og valideres korrekt.	ESP anmoder om data fra HTU-sensoren., og konverterer til det rigtige format.	OK
UC2-T2	ESP sender behandlet data til HTTP-serveren via POST.	Data gemmes midlertidigt på serveren.	Det skete som forventet	OK
UC2-T3	Display viser data fra sensorer.	Data præsenteres korrekt lokalt.	Lysintensitet, temperatur, fugtighed, strøm og IP-adressen fremvises til displayet.	OK
UC2-T4	Lyssensor leverer meget høje værdier (intensivt lys).	Systemet håndterer ekstreme værdier korrekt uden at fejle.		OK
UC2-T5	HTU21-sensor leverer ugyldige værdier (f.eks. -999 som temperatur).	ESP ignorerer disse værdier og logger fejlen.	Det skete som forventet	OK
UC2-T6	Data behandles til ukorrekt format, før det sendes til serveren.	Systemet detekterer fejl og genstarter dataindsamlingen.	Det skete som forventet	OK
UC2-T7	HTTP-serveren er offline under dataoverførsel.	Data gemmes lokalt på ESP, og genindsendelse forsøges senere.	Ikke opfyldt	NEJ

13.3 Hovedresultater

Projektgruppen har gennemført en accepttest med det formål at validere implementeringen af de opstillede krav. Testforløbet viste sig dog at være mere udfordrende end forventet. I forbindelse med UC1 blev flere tests ikke godkendt, hvilket var uventet. Derudover var det på forhånd forventet, at UC4 ikke ville blive godkendt, da systemet ikke blev færdiggjort inden for den tilgængelige tidsramme. Til gengæld levede mange af testene for UC2 og UC3 op til både de forventede observationer og de faktiske resultater, hvilket var positivt.

14 Diskussion af resultater

Under udviklingen af vores IoT-baserede Dual-Axis Solar Tracking System har vi opnået mange af de mål, vi satte os i starten, men vi stødte også på udfordringer, der gav os en masse at lære. Systemet fungerer overordnet godt og kan justere solpanelerne baseret på lysintensitet fra fire forskellige retninger. Det var vores hovedmål, og det lykkedes vi med ved at integrere lyssensorer og aktuatorer. Vi fik også lavet en webgrænseflade, hvor brugerne kan overvåge systemets data i realtid, hvilket opfyldte kravene til brugervenlighed.

Der var dog nogle krav, som vi ikke helt nåede at løse optimalt. For eksempel bruger systemet mere strøm, end vi havde håbet, især på grund af motorenes høje energiforbrug. Det betyder, at det i fremtiden kunne være en fordel at optimere strømstyringen for at gøre systemet mere bæredygtigt. Webgrænsefladen kunne også være mere stabil under høj belastning, da vi oplevede nogle problemer med ydeevnen, når der blev sendt mange forespørgsler.

En af de største udfordringer var præcisionen af lyssensorerne. De gav ikke altid helt nøjagtige aflæsninger, især under forskellige lysforhold. Det skyldtes blandt andet, at vi ikke havde en metode til at kompensere for omgivelseslys. Vi justerede softwaren for at mindske problemet, men det kan stadig forbedres med bedre kalibreringsmetoder. Derudover var motorenes præcision ikke perfekt – vi så små afvigelser i solpanelernes positionering, som formentlig skyldtes mekaniske tolerancer og begrænset feedback fra aktuatorerne. Det kunne løses i fremtiden ved at bruge en lukket kontrolsløjfe, der ville give mere præcis styring.

Vi er generelt tilfredse med designet og hvordan vi fik realiseret projektet. Kombinationen af ASE-modellen og Scrum-metoden fungerede godt for os. ASE-modellen hjalp os med at skabe struktur i starten, hvor vi udarbejdede kravspecifikation og design, mens Scrum gav os fleksibilitet til at justere projektet løbende. Det gjorde, at vi kunne arbejde effektivt og reagere på problemer, efterhånden som de opstod. Ved at bruge prototyping og testdreven udvikling kunne vi teste systemets funktion tidligt og undgå mange fejl senere i processen.

Når det er sagt, kunne vi have gjort nogle ting bedre. Vi testede hardware og software grundigt, men mest under kontrollerede forhold. Hvis vi havde haft tid til at teste systemet mere i realistiske omgivelser, kunne vi have fået et bedre billede af, hvordan det ville klare sig i praksis. Vi kunne også have arbejdet mere med at optimere softwaren for at gøre systemet hurtigere og mere stabilt.

Alt i alt har vi opnået et godt resultat, hvor de vigtigste krav blev opfyldt, og vi har lært en masse af de udfordringer, vi stødte på undervejs. Vi har fået værdifuld erfaring med at designe og bygge komplekse systemer, som vi helt sikkert kan bruge til at forbedre vores arbejde i fremtidige projekter. Der er stadig plads til optimering og forbedringer, men projektet har givet os et solidt fundament for fremtiden.

15 Konklusion og perspektivering

Projektet med udviklingen af et IoT-baseret Dual-Axis Solar Tracking System (IoT-DASTS) har været både udfordrende og lærerigt. Vi har opnået vores mål om at skabe et system, der kan justere solpanelers orientering baseret på realtidsdata fra lyssensorer. Systemet fungerer som forventet og demonstrerer præcision i motorstyring og sensordataindsamling. Derudover er webgrænsefladen blevet implementeret med succes, hvilket gør det muligt for brugere at overvåge og interagere med systemet på en enkel og intuitiv måde.

Gennem projektet har vi gjort brug af både ASE-modellen og Scrum som rammer for vores arbejdsproces. ASE-modellen gav os struktur i de indledende faser, hvor vi arbejdede med kravspecifikation og systemdesign, mens Scrum gjorde det muligt at arbejde agilt og tilpasse os løbende udfordringer og feedback. Kombinationen af disse metoder har vist sig at være effektiv og har sikret en velorganiseret proces.

Vi opfyldte de fleste krav til systemet, men der er stadig plads til forbedring. Strømforbruget er højere, end vi havde ønsket, og vi oplevede visse udfordringer med sensorernes nøjagtighed og motorenes præcision. Disse områder kan optimeres i fremtidige iterationer af projektet. Samtidig kunne testprocessen være blevet udvidet til mere realistiske miljøer for at sikre, at systemet

fungerer under forskellige forhold.

På et læringsmæssigt plan har projektet givet os erfaring med at arbejde i teams og anvende metoder til agil udvikling. Vi har styrket vores tekniske færdigheder inden for integration af hardware og software og har fået en dybere forståelse af, hvordan komplekse systemer designes og realiseres. Dette projekt har givet os et solidt fundament, som vi kan bygge videre på i fremtidige projekter.

Samlet set er vi forholdsvis tilfredse med resultatet af projektet, som ikke kun har opfyldt de fleste krav, men også har givet os vigtig indsigt og erfaring. Fremadrettet vil vi fokusere på at optimere systemets ydeevne, reducere strømforbruget og styrke robustheden, så systemet kan anvendes i større og mere komplekse sammenhænge. Projektet har vist os potentialet for bæredygtige løsninger gennem teknologi, og vi ser frem til at videreudvikle disse koncepter.

16 Bilagsliste

- Device.zip (koden)
- Process
- * Risk analyse
- * Gruppekontrakt
- * Metoder og arbejdsproces
- * Arbejdsfordeling

- Projekt
- * Arkitektur(bdd, ibd og domænemodel)
- * Design
- * Kravspecifikation(Kravspecifikation + use cases, Moscow analyse, FURPS, Aktør analyse, use case diagram.drawio, Accepttestspecifikation og Sekvens diagrammer)
- * Metode/proces(Forkortelse, logbog og problemformulering)

17 Referenceliste

- Datasheet til HT12D
<https://drive.google.com/drive/folders/1DGVtoIql0PP8A3auHBkJjwRtbPDXavX8>
- Datasheet til LDR
<https://drive.google.com/drive/folders/1DGVtoIql0PP8A3auHBkJjwRtbPDXavX8>
- Datasheet til t8_v1.7.1
<https://drive.google.com/drive/folders/1DGVtoIql0PP8A3auHBkJjwRtbPDXavX8>
- *Rapportskrivning*
<https://brightspace.au.dk/d2l/le/lessons/146528/topics/2023335>
- *TeknologiogRisikoanalyse*
<https://brightspace.au.dk/d2l/le/lessons/146528/topics/1966047>
- *Reviews*
<https://brightspace.au.dk/d2l/le/lessons/146528/topics/1966044>
- *Procesbriskrivelse*
<https://brightspace.au.dk/d2l/le/lessons/146528/topics/2023334>