Høgskulen på Vestlandet

Bacheloroppgave

ELE350

Predefinert informasjon

Startdato:	08-05-2023 09:00 CEST	**Termin:**	2023 VÅR
Sluttdato:	22-05-2023 14:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	Bacheloroppgave		
Flowkode:	203 ELE350 1 O 2023 VÅR		
Intern sensor:	Ilker Meric		

Deltaker

Navn:	Helene Holmgren Andersen
Kandidatnr.:	325
HVL-id:	589632@hvl.no

Informasjon fra deltaker

Egenerklæring *:	Ja
Inneholder besvarelsen konfidensielt materiale?:	Nei
Jeg bekrefter at jeg har registrert oppgavetittelen på norsk og engelsk i StudentWeb og vet at denne vil stå på vitnemålet mitt *:	Ja

Gruppe

Gruppenavn:	BO23EB-24
Gruppenummer:	29
Andre medlemmer i gruppen:	Stine Hopland

Jeg godkjenner avtalen om publisering av bacheloroppgaven min *

Ja

Er bacheloroppgaven skrevet som del av et større forskningsprosjekt ved HVL? *

Nei

Høgskulen på Vestlandet

BACHELOROPPGAVE:

BO23EB-24

Trådløs Temperaturregulering og Overvåking via BLE teknologi

Stine Hopland
Helene Holmgren Andersen

22. mai. 2023

Dokumentkontroll

Rapportens tittel:	*Dato/Versjon*
BO23EB-24 Trådløs Temperaturregulering og Overvåking via BLE teknologi	22. mai. 2023/2.0
	Rapportnummer: BO23EB-24
Forfatter(e): Stine Hopland Helene Holmgren Andersen	*Studieretning:* AUTB_2020 *Antall sider m/vedlegg* 69
Høgskolens veileder: Ilker Meric	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver:	*Oppdragsgivers referanse:*
Universitetet i Bergen, Institutt for Geovitenskap	Enver Alagoz
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Enver Alagoz* *enver.alagoz@uib.no*	

Revisjon	**Dato**	**Status**	**Utført av**
1.0	01.05.23	Første utkast	Stine Hopland & Helene H. Andersen
1.5	11.05.23	Utkast med kommentarer fra veileder	Ilker Meric
2.0	22.05.23	Endelig rapport	Stine Hopland & Helene H. Andersen

Forord

Denne rapporten er skrevet i forbindelse med vår avsluttende bacheloroppgave ved Høgskulen på Vestlandet (HVL), våren 2023. Oppgaven har gått ut på å lage et system for temperaturregulering via IoT-teknologi, samt utvikle tilhørende programvare slik at det kan benyttes av FARLAB på Universitetet i Bergen, Institutt for Geovitenskap.

Vi vil først og fremst takke Enver Alagoz, overingeniør ved Institutt for Geovitenskap og oppdragsgiver for denne bacheloroppgaven. Enver har bidratt med god hjelp og veiledning underveis, samt vist enormt engasjement i prosjektet, noe han fortjener en stor takk for. Vi vil også takke Steffen Brandt, vår medstudent fra UiB, som har bidratt med nyttige tilbakemeldinger ved testing av programvaren. Til slutt vil vi takke veilederen vår, Ilker Meric, som har sørget for jevnlig oppfølging gjennom hele arbeidet med oppgaven og kommet med gode tips og tilbakemeldinger.

Sammendrag

Denne bacheloroppgaven har bestått av å lage et system for overvåking og regulering av temperatur via Bluetooth, ved bruk av et brukergrensesnitt kompilert i Visual Studio. Systemet består av en datamaskin som kan kjøre programvaren vi har utviklet, en mikrokontroller med innebygget Bluetooth-modul som bidrar med kommunikasjon mellom hardware og software komponentene, samt en temperatursensor og et varmeelement.

Hensikten med oppgaven er å lage et digitalt system for overvåking og regulering av temperatur som fungerer sammen med instrumentene som brukes på massespektroskopi-laboratoriet FARLAB, som hører til Institutt for Geovitenskap på Universitetet i Bergen.

Prosjektet har i hovedsak gått ut på å utvikle programvaren som gjør det mulig å lese temperaturdata fra en temperatursensor, samt styre varmeelementet trådløst gjennom Bluetooth-kommunikasjon. I det ferdige programmet er brukergrensesnittet designet på en måte som lar brukeren enkelt koble til mikrokontrolleren og starte overvåking, samt velge ønsket temperatur for å sette i gang reguleringen. Programkoden er også skrevet slik at det enkelt skal kunne gjøres endringer for å tilpasses ulike bruksområder i fremtiden. Prosjektet resulterte i et program og et komplett system som oppfyller kravene gitt av oppdragsgiver.

Abstract

This bachelor thesis has involved creating a system for monitoring and regulating temperature using Bluetooth technology, with a user interface compiled in Visual Studio. The system consists of a computer capable of running the development software, a microcontroller with a built-in Bluetooth module for communication between the software and hardware components, as well as a temperature sensor and a heating element.

The purpose of the project has been to develop a digital system for monitoring and regulating temperature that can work with the instruments used in the FARLAB mass spectrometry laboratory, which belongs to the Department of Earth Science at the University of Bergen.

The project has primarily focused on developing the software that enables reading temperature data from a temperature sensor and wirelessly controlling the heating element through Bluetooth communication. In the finished program, the user interface is designed to allow users to easily connect to the microcontroller, initiate monitoring, and select the desired temperature for regulation. The program code is also written in a way that facilitates future modifications to adapt to different applications. Overall, this project has resulted in a program and a complete system that meet the specified requirements.

Innholdsfortegnelse

1 Innledning

1.1 Organisering av rapporten

En betydelig mengde av arbeidet med dette bachelorprosjektet har vært programmering og programvareutvikling, samt kobling av ulike hardware komponenter. Det ville vært overveldende dersom programkoden og alle de tekniske spesifikasjonene ble presentert i hoveddelen av rapporten, og derfor står de utfyllende detaljene i appendiks.

Vi bruker IEEE sin standard til kildehenvisning, hvor kildene er markert med firkantparanteser, slik [x]. Forkortelser, begrep og andre ord som vi mener trenger videre forklaring vil ligge i Appendix A.

1.1.1 Hovedkapitlene

Innledningsvis i denne rapporten vil vi introdusere vår oppdragsgiver og gi en beskrivelse av problemstillingen for oppgaven. Deretter vil vi bryte ned kravspesifikasjonen for oppgaven, diskutere problemstilling og legge frem løsningsforslag for oppgaven. Videre introduserer vi viktige begreper og informasjon knyttet til Bluetooth Low Energy (BLE) teknologi, før vi presenterer ferdig løsning og testing av løsningen. Til slutt vil vi oppsummere prosessen med prosjektarbeidet, samt gi forslag til fremtidig arbeid og avslutte med konklusjon.

1.1.2 Appendiksene

I appendiksene vil man finne brukerdokumentasjon og utfyllende informasjon om software som ble utviklet i forbindelse med dette prosjektet. Der finnes også alle tekniske detaljer knyttet til hardware-komponentene som er benyttet. I tillegg finner man forkortelser og ordforklaringer, figur-liste, informasjon om prosjektorganisering og fremdriftsplan, samt risikoliste og liste over materialer.

Kildekoden er tilgjengelig på GitHub-repositoriet[1] "BluetoothTemperatureRegulation".

1.2 Oppdragsgiver

Oppdragsgiver for denne bacheloroppgaven er Universitetet i Bergen (UiB), Institutt for Geovitenskap. UiB ble grunnlagt i 1946, mens institutt for geovitenskap ble etablert 1. januar 2003. Instituttet holder til i Realfagsbygget på Nygårdshøyden i Bergen, og har nesten 80 ansatte og mer enn 70 stipendiater. Satsningsområdene for deres forskning er energi, klima, miljø, ressurser og naturkatastrofer [1].

FARLAB er et massespektroskopi-laboratorium ved Institutt for Geovitenskap, hvor de måler isotoper av ulike grunnstoff, samt isotopklumping i karbonater [2]. Målet til FARLAB er å kunne adressere viktige problemstillinger innenfor geovitenskap, ved å analysere disse isotopene. Vår kontaktperson på UiB er

[1] https://github.com/stinehopland/BluetoothTemperatureRegulation

forsker og jobber på FARLAB, og denne oppgaven skal ta for seg temperaturovervåking og regulering for noen av instrumentene på laboratoriet.

1.3 Problemstilling

Instrumentene på FARLAB, inkludert MAT253, MAT253Plus og Kiel IV, brukes til å analysere partikler og polymer, samt undersøke termiske egenskaper til ulike materialer [3] [4]. Disse instrumentene muliggjør grundige analyser i laboratoriet, blant annet ved å ta i bruk kuldefeller som fjerner forurensninger fra prøvegassen [5]. Temperaturen på kuldefellene kontrolleres ved hjelp av mikrokontrollbaserte temperaturregulatorer, inkludert bruk av PID-regulering. Under massespektroskopi vil prøvene måtte utsettes for temperaturer mellom -50 °C og +150 °C.

Det eksisterende systemet for temperaturregulering har per i dag et analogt brukergrensesnitt, og regulatoren kan ikke styres via en datamaskin. Dersom prøvene blir eksponert for feil temperatur, ødelegger det resultatene og prøvene må kastes. Det koster mye penger å drifte instrumentene, og dyrebare prøver går til spille. På grunn av dette ønsker UiB å oppgradere til en digital løsning slik at man raskt kan se om temperaturen er feil, og kan følge med på hva temperaturen har vært under hele prosessen. Figur 1 viser det fysiske oppsettet av instrumenter til prosessering av prøver på FARLAB.

Figur 1: Fysisk oppsett av instrumenter brukt til prosessering av prøver på lab (Bilde tatt i FARLAB ved Universitetet i Bergen 27.01.2023)

Figur 2: Eksisterende temperaturkontroller (Bilde tatt i FARLAB ved Universitetet i Bergen 27.01.2023)

Figur 2 viser den nåværende temperaturkontrolleren som er brukt på FARLAB. Det er en PID-regulator der ønsket temperatur stilles inn manuelt uten mulighet for overvåking av faktisk temperatur underveis.

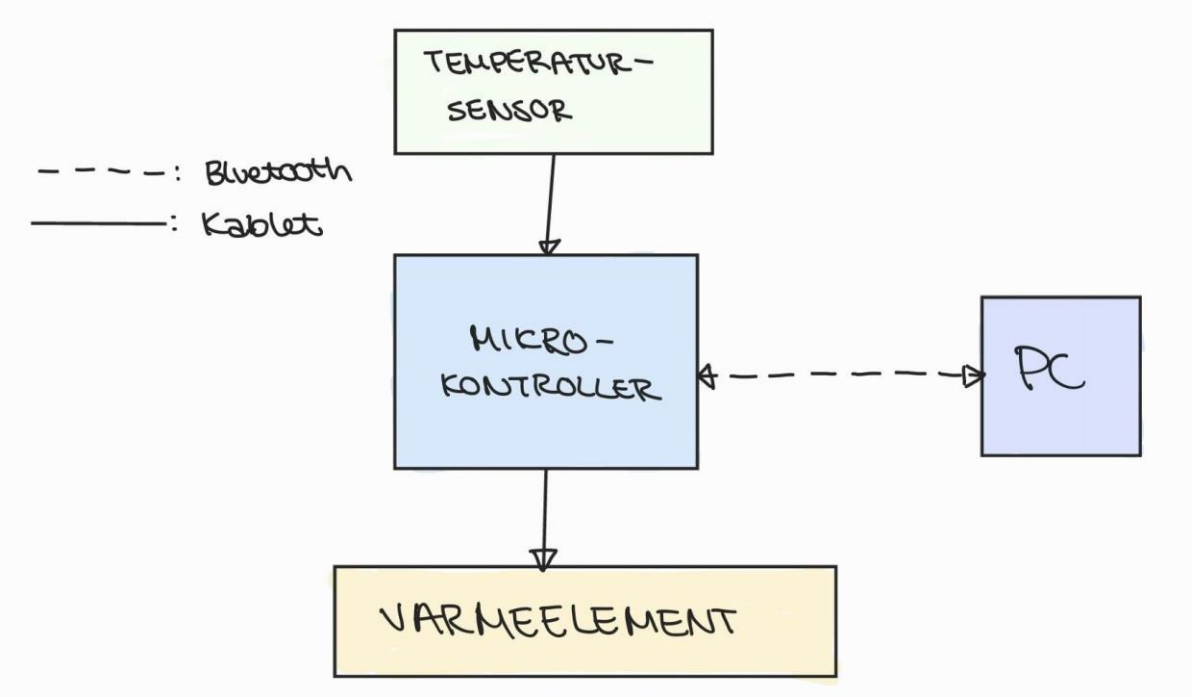
Figur 3: Bilde av datamaskiner på og instrumenter på lab (Bilde tatt i FARLAB ved Universitetet i Bergen 03.02.2023)

Som man ser på figur 3 er det stasjonære datamaskiner plassert like ved oppsettet av instrumenter og disse vil brukes til å overvåke temperaturen med det nye, digitale systemet vi skal utvikle.

1.4 Hovedidé for løsningsforslag

Målet er å oppgradere det eksisterende temperaturovervåkingssystemet til å bruke IoT-løsninger, slik at overvåking og regulering av temperaturen kan foregå digitalt og trådløst. Oppdragsgiver ønsker et system styrt av en mikrokontroller som kommuniserer trådløst med en datamaskin og gir kontinuerlig overvåking og regulering av temperatur. På sikt ønsker oppdragsgiver at temperaturovervåkningen skal kunne deles med flere datamaskiner og være tilgjengelig for alle som trenger tilgang til det over nett eller skylagring. Dette er derimot ikke mulig den dag i dag, grunnet sikkerhetsretningslinjer satt av IT-avdelingen til UiB. Av den grunn vil vi bruke Bluetooth teknologi for å overføre målingene til én enkelt datamaskin i vår løsning, fremfor å bruke trådløst nettverk.

I den endelige løsningen ser vi for oss et fullstendig system for overvåking- og regulering av temperatur ved bruk av et brukergrensesnitt på PC. Det skal kunne kjøres kontinuerlig 24 timer i døgnet, og bruke Bluetooth-kommunikasjon for å overføre data trådløst mellom datamaskinen og mikrokontrolleren.



Figur 4: Skisse over system for regulering og overvåking av temperatur.

Figur 4 viser en enkel skisse av hvordan systemet vil bygges opp. Reguleringssystemet skal bestå av en temperatursensor og et varmeelement som styres av mikrokontrolleren via et solid state relé. Mikrokontrolleren leser av temperaturverdier fra temperatursensoren og sender målingene over til datamaskinen. Programmet som kjøres på datamaskinen sammenligner målt temperatur med angitt temperatur satt av bruker, og sender beskjed til mikrokontrolleren dersom varmeelement må skrus av eller på.

Brukergrensesnittet skal være enkelt utformet, gi en oversikt over temperaturmålingene og lagre målingene lokalt på datamaskinen. Vårt mål for denne oppgaven er å lage et fleksibelt temperaturovervåking- og reguleringssystem som skal kunne modifiseres og endres slik at det kan brukes til mange ulike instrumenter og prosjekter i fremtiden.

2 Kravspesifikasjon

I dette kapittelet listes kravene til prosjektet i fire underkategorier: krav til temperaturmåling, krav til temperaturregulering, krav til brukergrensesnitt og krav som omhandler hardware-komponenter.

2.1 Liste over kravspesifikasjon

Krav til temperaturregulering:

- Temperatur skal holdes innenfor +/- 1 °C fra valgt settverdi
- Regulering skal gjøres ved bruk av brukergrensesnitt

Krav til temperaturmåling:

- Mikrokontroller leser av data fra temperatursensoren hvert sekund
- Data fra temperatursensor videresendes trådløst til datamaskin fra mikrokontroller
- Programkoden for mikrokontroller må være oversiktlig og godt kommentert slik at den enkelt kan endres/modifiseres etter behov

Krav til grafisk brukergrensesnitt kompilert i Visual Studio:

- Windows Forms og C# programmeringsspråk
- Ønsket/forventet temperatur skal oppgis
- Inneholde Start- og Stopp-knapper
- Display som viser temperatur per tid i graf
- Lagre målingene i txt-filer på en lokal datamaskin
- Kan kjøres kontinuerlig
- Motta målinger trådløst fra mikrokontroller via Bluetooth
- Engelsk

Krav som omhandler hardware-komponenter

- Arduino Nano 33 BLE som mikrokontroller
- SHT31 temperatursensor

3 Analyse og løsning for problemstilling

Vi vil i dette kapittelet først analysere problemstillingen til oppgaven, for å så komme med en utforming av løsning basert på analysen. Deretter vil vi beskrive noen refleksjoner vi gjorde rundt valg av hardware-komponenter.

3.1 Analyse

Målet med oppgaven er å lage et enkelt program som både regulerer og overvåker temperatur ved bruk av et brukergrensesnitt på en lokal datamaskin. Mikrokontrolleren som er tilkoblet temperatursensoren og varmeelementet skal kunne kommunisere trådløst med datamaskinen ved

bruk av Bluetooth-kommunikasjon. Reguleringen skal foregå ved at varmeelement skrus på når temperaturen ligger én grad under gitt settverdi, og skrus av når målingen viser én grad høyere enn settverdien. Denne løsningen skal erstatte PID-kontrolleren som brukes per dags dato. Så lenge temperaturen holdes i nærheten av ønsket temperatur skader ikke dette prøvene, og det er derfor tilstrekkelig å regulere ved å skru av og på varmeelementet.

I vår løsning skal vi benytte mikrokontrolleren Arduino Nano 33 BLE (Se Appendiks E.1), og vil derfor bruke Arduino IDE for å programmere denne (Se Appendiks C.1). For at temperaturreguleringen skal være effektiv, er det nødvendig å lese av temperaturen fra temperatursensoren så ofte som hvert sekund, slik at temperaturen ikke kommer for høyt opp eller for lavt ned før man leser av temperatur igjen.

I starten av prosjektet vil vi også lagre temperaturmålingene hvert sekund for å enkelt kunne se og teste om systemet fungerer. I fremtiden vil det derimot ikke være behov for å lagre målingene like ofte ettersom temperaturen overvåkes i lengre perioder, ofte over flere døgn. Da blir det mer aktuelt å lagre temperaturen én gang hvert 5. minutt eller sjeldnere, så dette intervallet må enkelt kunne endres i programmet. Koden må være oversiktlig, ryddig og godt dokumentert slik at eventuelle fremtidige brukere enkelt kan gjøre endringer.

Lagring av målinger skjer via datamaskinen vi kjører brukergrensesnittet fra, og lagres lokalt i tekstfiler. Det vil utføres målinger knyttet til ulike prøver, så det må være mulig å opprette nye tekstfiler knyttet til de ulike øktene. Målingene skal inneholde målt temperaturverdi, ønsket temperaturverdi, samt dato og tidspunkt for målingen.

Mange av de som jobber ved UiB og FARLAB har ikke norsk som morsmål og det er derfor viktig at brukergrensesnittet kommuniserer på engelsk slik at alle som skal bruke programmet kan forstå det. Det er også en fordel å skrive programkoden på engelsk slik at koden kan brukes og modifiseres av andre i fremtiden som har lignende prosjekter.

Ettersom Arduino skal kommunisere trådløst med datamaskin trenger den egen strømforsyning. Temperaturreguleringen blir en del av en fast installasjon som har tilgang til strømuttak og vi er ikke avhengig av at reguleringssystemet skal være portabelt. Det vil derfor være naturlig å koble Arduino til et strømuttak dersom dette er mulig.

3.2 Utforming av løsningen

Hovedformålet med denne oppgaven er å utvikle IoT-teknologi for temperaturovervåking som skal erstatte det eksisterende systemet for regulering av temperatur. Oppdragsgiver har lagt frem tydelige rammer for løsningen, dermed har vi ikke vurdert mange ulike løsningsalternativer. Oppgavebeskrivelsen spesifiserte hvilke utstyr vi skulle bruke og beskrivelse av en ønsket løsning, samt foreslått program for grafisk brukergrensesnitt. Detaljert informasjon om hardware-komponentene og koblingsskjema for disse finnes i Appendiks E.

Som nevnt tidligere, skal vi bruke mikrokontrolleren Arduino Nano 33 BLE og Visual Studio som utviklingsmiljø. Oppgaven nevnte i utgangspunktet bruk av en PT100 som temperatursensor, men oppdragsgiver ønsket at vi heller skulle benytte en SHT31 temperatursensor under utvikling og testing av programvaren. Visual Studio hadde vi allerede installert med lisens, og oppdragsgiver hadde varmeelement tilgjengelig på FARLAB, så vi trengte kun å gå til innkjøp av Arduino og temperatursensor.

I den endelige løsningen brukes Arduino til å lese av temperaturmålinger fra temperatursensor, og skru av og på varmeelement for å regulere temperatur. Vi bruker Visual Studio og programmeringsspråket C# for å kompilere brukergrensesnitt som brukes til overvåking av temperatur og styring av temperaturregulering. Programmet skal ha mulighet til å sette ønsket temperatur, kjøre varmeelementet, måle temperatur og slå av varmeelementet når settverdien er nådd. Dersom temperaturen går noen grader over eller under ønsket temperatur er ikke det noe problem ifølge oppdragsgiver. Det holder derfor å skru av varmeelementet når ønsket temperatur er nådd, og skru på igjen varmeelementet når temperaturen er én grad lavere enn ønsket temperatur. Temperaturmålingene lagres i tekstfiler på en lokal datamaskin. Målingene plottes også i en graf i brukergrensesnittet slik at man enkelt kan observere temperaturen over tid og oppdage eventuelle avvik.

3.2.1 Vurderinger i forhold til verktøy og HW/SW komponenter

Når det kommer til mikrokontrollere, har vi erfaring med Arduino Uno gjennom faget ELE102 - Programmering og Mikrokontrollere på HVL fra før. Arduino Nano 33 BLE ble valgt for å ha mulighet til å kommunisere med en datamaskin via Bluetooth. Raspberry Pi ble vurdert av oppdragsgiver som en mulig mikrokontroller, men valget endte på Arduino fordi det er en enklere løsning og den er god nok for det vi skal bruke den til. De eneste oppgavene til Arduinoen er å kommunisere med datamaskinen og å styre de digitale utgangene. Raspberry Pi har mye mer regnekraft, og fungerer som en liten datamaskin. Dette er unødvendig komplisert for vår oppgave. Arduino Nano 33 BLE må kun kobles til datamaskinen én gang for å få lastet ned programkoden og kjører deretter av seg selv.

Arduino skal regulere varmeelement via et solid state relé. Solid state reléet brukes som en bryter for å skru av og på varmeelementet. Varmeelementet krever høyere spenning (230V), men ved å bruke et solid state relé kan Arduino skru det av og på med bare 3 V spenning (se Appendiks E.3).

Temperatursensoren vi bruker i denne oppgaven er Grove sin SHT31. Denne tolererer temperaturer fra -40 til 125 °C (Se Appendiks E.2). Som nevnt i problemstillingen skal instrumentene på FARLAB behandle prøver i temperaturer fra -50 til 150 °C, og dermed vil systemet trenge en annen temperatursensor med større spenn når det faktisk skal brukes med disse instrumentene. Etter hvert vil sensoren vi bruker antagelig bli erstattet av en PT100, ettersom den tolererer både høyere og lavere temperaturer enn SHT31. Hovedårsaken til at oppdragsgiver ønsket at vi benyttet SHT31 er at Arduino henter digital data direkte fra SHT31 sensoren gjennom Arduino biblioteket som hører til sensoren (se kapittel 5.1.1). En PT100 derimot er en resistiv temperatursensor som måler endringer i motstand basert på temperatur og vil derfor kreve justering av både kode og krets for å konvertere den resistive verdien til et målbart signal [7]. Oppdragsgiver er klar over forskjellene, og vet hvordan de skal gjøre om på koden og koblingen så vi vil ikke ha fokus på dette videre i oppgaven.

Dataprogrammet som skal styre hele systemet valgte vi å skrive i C# ved hjelp av Visual Studio 2022. Dette programmeringsspråket og utviklingsmiljøet har vi erfaring med fra fag vi har tatt på HVL i løpet av utdannelsen. Visual Studio har gode verktøy for å lage GUI program, og ble dermed et naturlig valg for oss i denne oppgaven. Det er også kompatibelt med Windows, som er det operativsystemet som blir brukt på datamaskinene i laboratoriene temperaturovervåkningen skal brukes i.

Når det kommer til strømforsyning av Arduino har vi to alternativer: enten bruke micro-USB kabel og en USB-strømadapter for kontinuerlig strømtilførsel eller bruke et batteri som strømkilde [8]. Som vi allerede har kommet frem til i kapittel 3.1. er det naturlig for oss å velge det første alternativet. Strømkontakten vil gi 5 V spenning gjennom micro-USB-porten på Arduino brettet, og den interne spenningsregulatoren vil da regulere dette ned til 3.3 V som Arduino Nano 33 BLE opererer med (se Appendiks E.1). Slik vil man være sikret at mikrokontrolleren alltid har strøm 24 timer i døgnet, og man trenger ikke å bekymre seg for å måtte skifte batteri. Ulempen med dette er at dersom strømmen i bygget eller rommet går vil Arduino kobles ut, men det vil også den tilkoblede PC-en, så vi ser ikke på det som nødvendig at Arduino skal fortsette å kjøre. Dataene lagres kontinuerlig i tekstfil, så når systemet er oppe og går igjen vil man kunne finne ut når målingene sluttet ved å se på de lagrede dataene. Arduino kunne også kobles til for eksempel et 9 V batteri via VIN-pin (Voltage IN) på Arduino brettet, men ettersom input spenningen på pinsene er 3.3 V, trenger vi en ekstern spenningsregulator for å regulere spenningen ned fra 9 V til 3.3 V (se Appendiks E.1). Dette er som nevnt ikke nødvendig, dersom vi bruker den designerte USB-porten, som videre understreker at det er det beste alternativet.

4 Bluetooth Low Energy

For å kunne lage et program med trådløs dataoverføring ved bruk av Bluetooth Low Energy (BLE) som den valgte mikrokontrolleren vår bruker, måtte vi sette oss inn i hvordan BLE teknologi fungerer. Dette er en såpass grunnleggende del av oppgaven, med terminologi som vil gå igjen gjennom hele oppgaven, så før vi beskriver løsningen vår starter vi med å introdusere de sentrale begrepene for BLE teknologi, og hvordan teknologien fungerer.

4.1 Spesifikasjoner

Bluetooth Low Energy er en trådløs kommunikasjonsteknologi som ble introdusert av Bluetooth Special Interest Group (SIG) i 2010 [9]. BLE er en nyere versjon av Bluetooth teknologi som supplerer den opprinnelige Bluetooth Classic teknologien, og er en alternativ måte å utveksle data trådløst. Bluetooth Classic støtter punkt-til-punkt kommunikasjon mellom enheter, og brukes i hovedsak til dataoverføring og som standard radioprotokoll for trådløse høyttalere, hodetelefoner og annen trådløs lydstrømming. BLE overfører data i det samme frekvensbåndet som Bluetooth Classic, men har ekspandert fra å bare ha punk-til-punkt kommunikasjon til å også støtte både kringkasting og mesh. BLE teknologi kan også bestemme avstand og retning av enheter og kan brukes til enhetsposisjoneringsteknologi. Som det ligger i navnet til teknologien, Bluetooth Low Energy, er denne mer energieffektiv enn Bluetooth Classic. BLE ble nemlig i utgangspunktet utviklet for å distribuere energiforbruket mellom enheter, slik at den enheten med størst batterikapasitet tar det meste av «jobben» med overføring av data. På den måten kan den andre enheten drives av et lite batteri, som igjen gjør at enheten kan være liten i størrelse [10].

Bluetooth Low Energy består av tre hoveddeler: Bluetooth Core Specification, Profiles & Services.

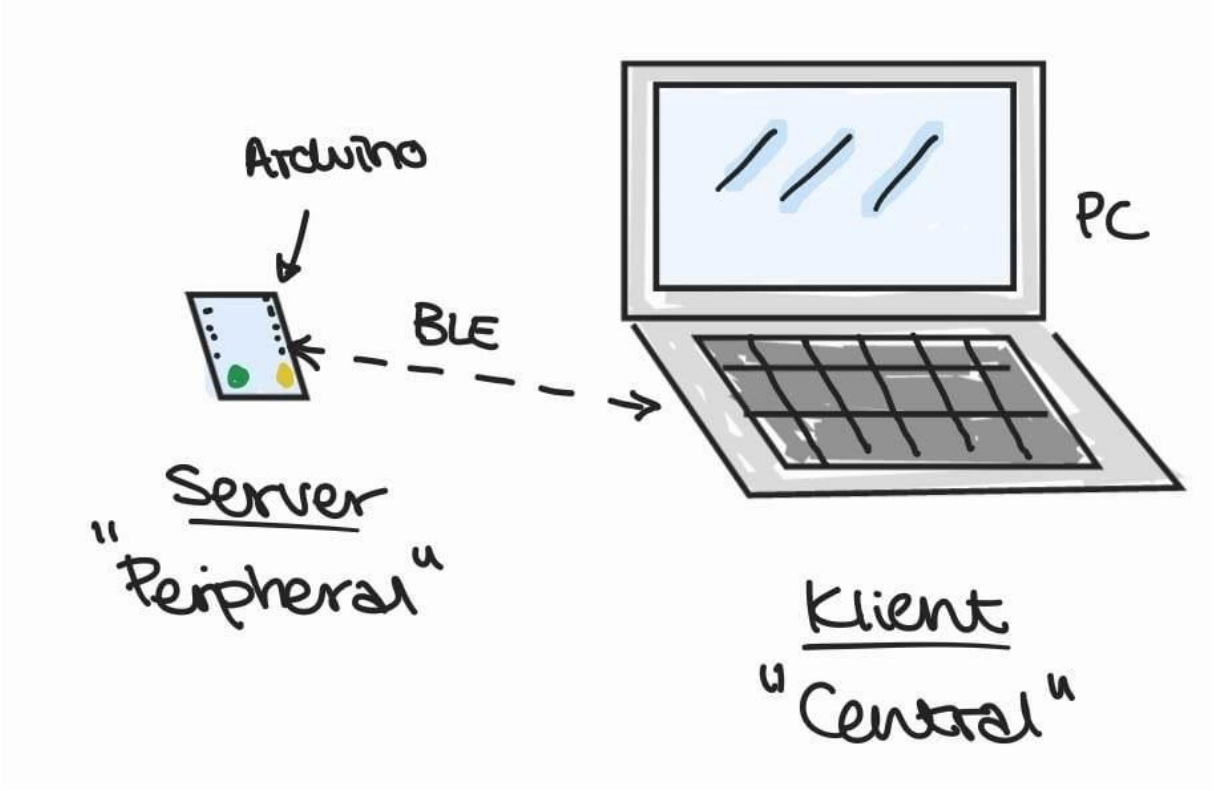
4.1.1 Bluetooth Core Specification

Bluetooth Core Specification er en teknisk standard som beskriver den teknologien som tillater kommunikasjon mellom Bluetooth enheter. Den omfatter detaljerte spesifikasjoner for Bluetooth-protokollen og definerer hvordan enheter skal kommunisere og koble seg sammen [10]. Bluetooth Core Specification er forhåndsdefinert på alle enheter med en Bluetooth-modul og er ikke noe vi skal endre på, men heller forholde oss til når vi programmerer.

4.1.2 Profiles

Når to BLE enheter kommuniserer med hverandre, vil det være et klient/server forhold mellom dem. Profil-spesifikasjonene er det som definerer rollene enhetene påtar seg, og hver BLE enhet må defineres som enten en klient eller en server.

Servere inneholder tilstandsdata, mens klienter bruker denne dataen på en eller annen måte. I tillegg er det vanligvis server-enheten som «annonserer» seg som en BLE enhet, og klienten som deretter etablerer en tilkobling [10]. I vår oppgave vil dette si at mikrokontrolleren (Arduino Nano 33 BLE) fungerer som en server, fordi den leser av temperaturer fra temperatursensoren og inneholder dataene vi som brukere ønsker å hente ut. Datamaskinen fungerer som klient og brukes for å hente ut data fra annen Bluetooth-enhet.



Figur 5: Illustrasjon Klient/Server forhold i henhold til bacheloroppgaven

Figur 5 viser en illustrasjon av klient/server forholdet som definert i denne oppgaven. Server er kalt «Peripheral» på engelsk, og det er dette navnet som brukes når man programmerer med Bluetooth-biblioteker. For klient bruker man det engelske ordet «Central».

4.1.3 Services

Tilstandsdataene som ligger på serveren er formelt definerte dataelementer, kjent som «characteristics» og «descriptors», men som i denne oppgaven vil refereres til som karakteristikker og beskrivelser. Disse er gruppert i konstruksjoner kjent som «services», og disse vil vi referere til som servicer i denne oppgaven. En servicespesifikasjon definerer altså en enkelt service, i tillegg til de karakteristikkene og beskrivelsene som den inneholder, og kan tenkes på som definisjonen av ett aspekt av serverenhetens oppførsel [10].

Når vi snakker om servicer, karakteristikker og beskrivelser innenfor BLE, er det viktig å inkludere deres universelle, unike identifikator som er det vi kaller UUID. Hver service, karakteristikk og beskrivelse har altså en UUID adresse som enten er egendefinert, eller en kjent identifikator definert av Bluetooth SIG som ligger i deres liste av «assigned numbers» [11]. Kjente UUID identifikatorer er 16-bit, og brukes for standardiserte servicer, karakteristikker og beskrivelser som for eksempel batteri informasjon eller pulsmålinger. En egendefinert UUID derimot er 128-bit, defineres av utvikleren, og brukes for karakteristikker som ikke er standardisert av Bluetooth SIG.

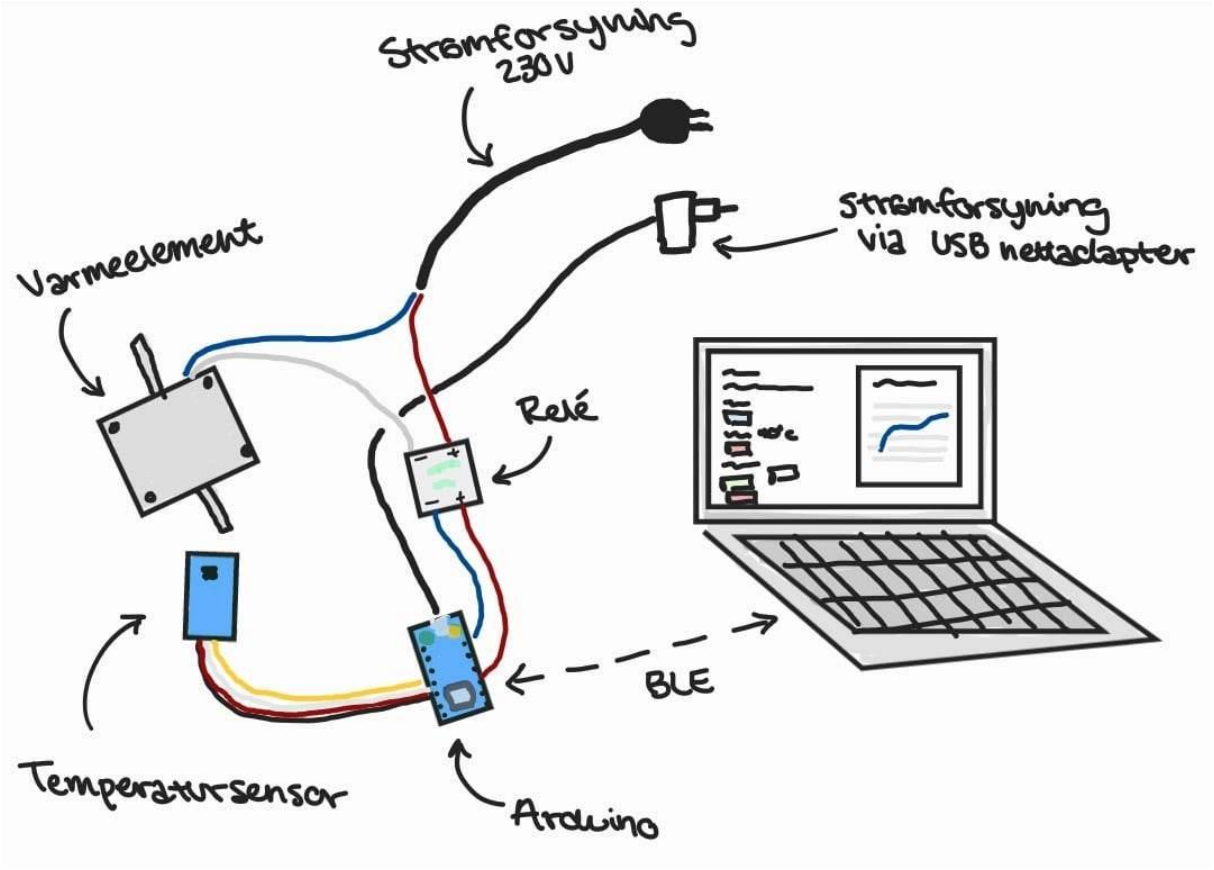
Til vår oppgave fant vi en SIG-definert standard UUID som passet til karakteristikken som beskriver temperaturdata som overføres via Bluetooth. Karakteristikken vi valgte har UUID'en «2A6F» og har navnet «Temperature Measurement» [11]. Vi fant ingen standard UUID som passet å bruke til verken karakteristikk for å skru av og på varmeelement eller service, så for disse valgte vi å lage egendefinerte UUID adresser. Dette gjorde vi gjennom nettsiden https://www.uuidgenerator.net/ som genererer unike og tilfeldige UUID adresser [22].

5 Realisering av valgt løsning

Dette kapittelet tar for seg den endelige løsningen til prosjektet. Vi begynner kapittelet med en kort beskrivelse av den helhetlige løsningen. Deretter går vi gjennom hver enkelt del hver for seg. Vi begynner med valgt mikrokontroller og program for denne, videre til brukergrensesnitt og sekvensdiagrammer som beskriver de viktigste elementene av programkoden.

5.1 Det fullstendige systemet

I figur 6 under kan du se en skisse av det ferdige systemet med alle komponenter og kobling mellom disse.



Figur 6: Skisse av ferdig løsning med alle komponenter

Temperaturreguleringssystemet fungerer slik at Arduino leser og dekoder temperaturdata fra sensoren (se kapittel 5.1.1), og sender denne dataen via BLE til datamaskinen som kjører applikasjonen vi har laget. Temperaturdataen blir lagret i en lokal tekstfil, samt vist i en graf så lenge Arduino er tilkoblet via BLE. Brukeren av applikasjonen har mulighet til å sette temperaturen til ønsket verdi, og programmet vil fortelle Arduino at varmeelementet må slås av eller på avhengig av om temperaturen er for høy eller lav i forhold til settverdien. Denne kommunikasjonen foregår også via BLE.

Regulering og overvåking foregår via brukergrensesnitt programmert i C# og kompilert i Visual Studio. Vi planla opprinnelig å lage et GUI-program i Windows Forms når vi skulle lage brukergrensesnitt, men vi endte opp med å heller benytte Universal Windows Platform (UWP). Dette valget ble tatt fordi vi var avhengig av å bruke et Bluetooth-bibliotek som kun var kompatibelt med UWP. I tillegg har UWP den fordelen at programmet kan lastes ned som en applikasjon på de datamaskinene som skal ta programmet i bruk. I utviklingen av programvaren benyttet vi en eksempelkode fra Microsoft som et utgangspunkt for vår egen programkode [12]. Denne eksempelkoden er en UWP applikasjon som bruker det overnevnte Bluetooth-biblioteket til å søke opp og koble til BLE enheter innen rekkevidde, og lar bruker se enheten sine annonserte servicer og karakteristikker.

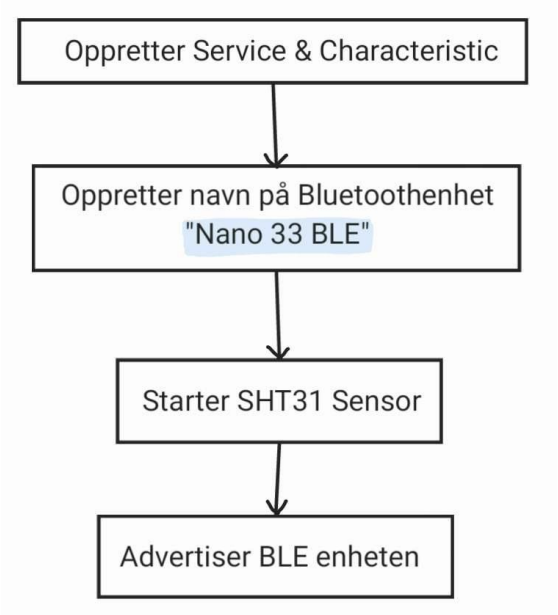
I utviklingen av vår egen applikasjon, har vi beholdt de metodene som søker etter BLE enheter, men gjort store endringer på resten av koden samt erstattet den delen av koden som står for selve brukergrensesnittet. I den ferdige applikasjonen har vi skrevet koden slik at programmet utelukkende søker etter BLE enheter med navnet «Nano 33 BLE» (Arduino) og kobler seg til denne, samt starter å lese av temperatur ved bare ett tastetrykk. UWP applikasjonen blir beskrevet videre i kapittel 5.2.

5.1.1 Arduino Program

Dette kapittelet beskriver hvordan Arduino programmet fungerer og hva det brukes til. Det fullstendige programmet kan leses i Appendiks D.1. Koden ligger også i vårt GitHub-repository[2].

Programmet bruker ArduinoBLE bibioteket, som er et bibliotek med innebygde BLE funksjoner som gjør at vi kan utnytte Bluetooth modulen til Arduino. Vi bruker også SHT31-biblioteket, som kommuniserer med sensoren gjennom seriell kommunikasjon og leser rådataene som representerer temperatur. Rådataene leses som 16-bits heltall, og biblioteket inneholder formler som brukes for å konvertere rådataen til en temperaturverdi. På den måten kan vi benytte de innebygde funksjonene til å få programmet til å skrive ut en temperaturverdi i celsius.

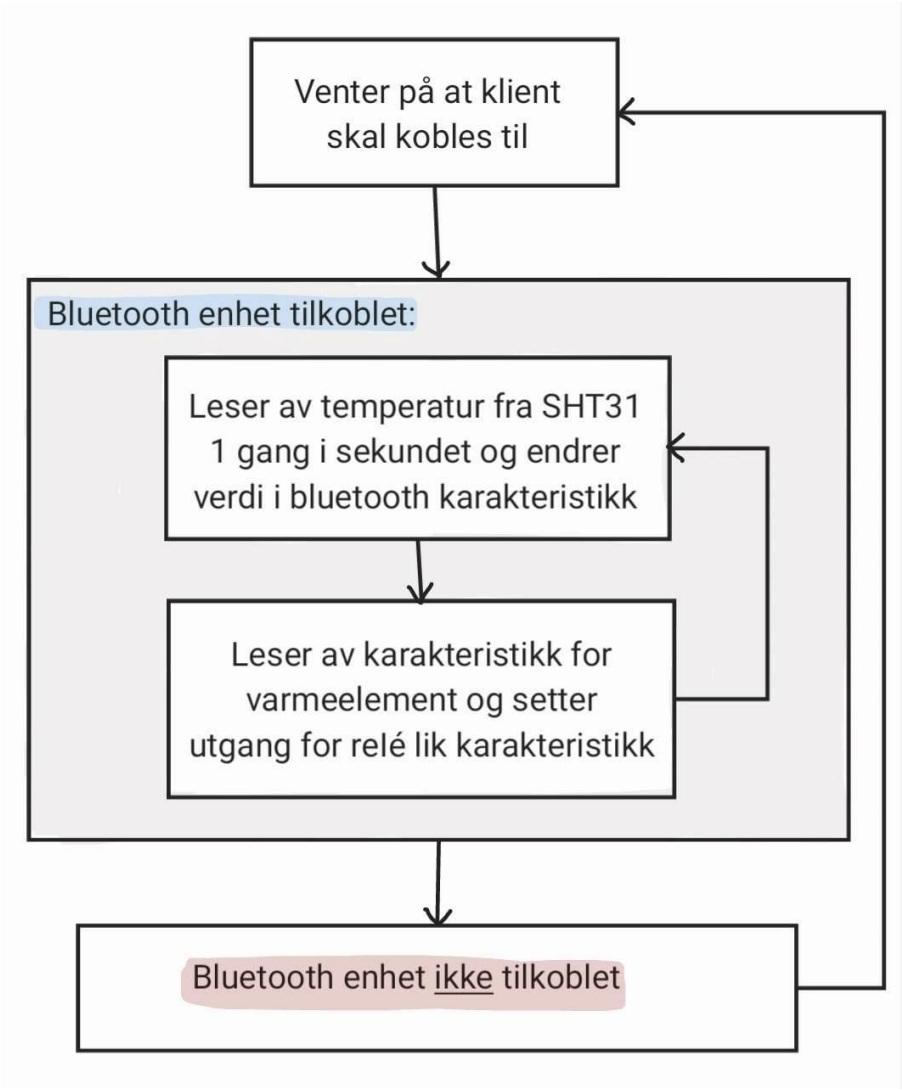
Arduino kode er delt opp i to deler: Setup og Loop. Setup kjøres kun en gang når mikrokontrolleren starter opp. Deretter går den over i Loop delen som kjøres kontinuerlig til man eventuelt kobler fra strømmen.



Figur 7: Oversikt over Setup for Arduino programkode

[2] https://github.com/stinehopland/BluetoothTemperatureRegulation

Figur 7 viser en oversikt over Setup delen av koden til Arduino. Programmet starter ved å sette opp PIN-utganger for varmeelementet og det innebygde LED-lyset, samt å sette opp mikrokontrolleren som en BLE server. Deretter annonserer programmet servicen til enheten, som inneholder de to karakteristikkene for henholdsvis temperatursensoren og varmeelementet. Disse karakteristikkene kan nå benyttes av en klient til å lese av temperaturmålingene eller slå varmeelementet av/på ved å kommunisere med server enheten over BLE.



Figur 8: Oversikt over Loop i Arduino programkode

Figur 8 viser en oversikt over hva som skjer i loopen til Arduino-koden. Programmet venter på at en klient skal koble seg til serveren. Når en klient er tilkoblet slås det innebygde LED-lyset seg på for å indikere en ny tilkobling. Programmet vil også begynne å lese temperaturdata fra temperatursensoren og sende det til karakteristikken for temperaturmålinger. Programmet leser også av verdien til varmeelement-karakteristikken og slår varmeelementet på eller av i henhold til hvilke verdier som ble lest av. Dette skjer hvert sekund så lenge programmet kjører, for å unngå å belaste mikrokontrolleren

mer enn nødvendig. Når klienten ikke er tilkoblet lenger, slår programmet av varmeelementet og LED-lyset og venter på en ny tilkobling.

Slik programmet er satt opp er det ingen funksjoner som fanger opp om temperatursensor eller varmeelement ikke er koblet til eller ikke fungerer som de skal. Arduino vil prøve å lese av temperaturen og skru av og på utgang til varmeelement som styrt fra brukergrensesnittet i Visual Studio, men målingen fra temperatursensor vil eventuelt bli sendt som en «ikke gyldig» verdi og vises som «NaN» (Not a Number) i overvåkningen i brukergrensesnittet. Dersom det er noe galt med tilkoblingen til, eller selve varmeelementet, vil dette kun kunne oppdages av bruker gjennom overvåkningen av temperatur i brukergrensesnittet ved at temperaturen ikke øker selv om det står at varmeelement er skrudd på, eller at temperaturen bare fortsetter å stige når det står at varmeelementet er skrudd av.

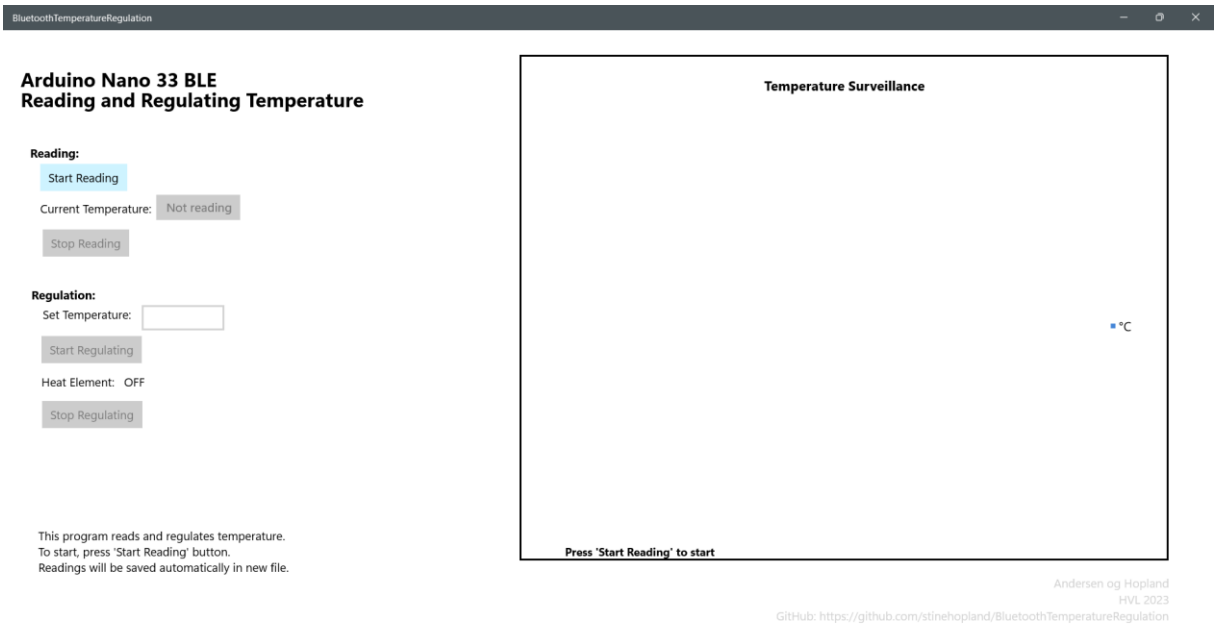
5.2 UWP applikasjon og brukergrensesnitt

Dette kapittelet tar for seg hvordan GUI programmet som kjøres på PC ser ut, samt hvordan det benyttes for temperaturovervåking og regulering. Den fullstendige programkoden er tilgjengelig i vår GitHub-repository[3].

5.2.1 Brukergrensesnitt og lagring av temperaturmålinger

I utviklingen av programmet har vi prøvd å legge så mye vekt på brukervennlighet som mulig. Målet har vært å lage en applikasjon med færrest mulig knapper og et oversiktlig display med temperaturmålinger. Tanken er at det skal være lett å forstå og intuitivt å bruke. For å oppnå dette, har vi utviklet programmet slik at det er mye som skjer på «bak-enden» av programmet som brukeren ikke ser.

[3] https://github.com/stinehopland/BluetoothTemperatureRegulation

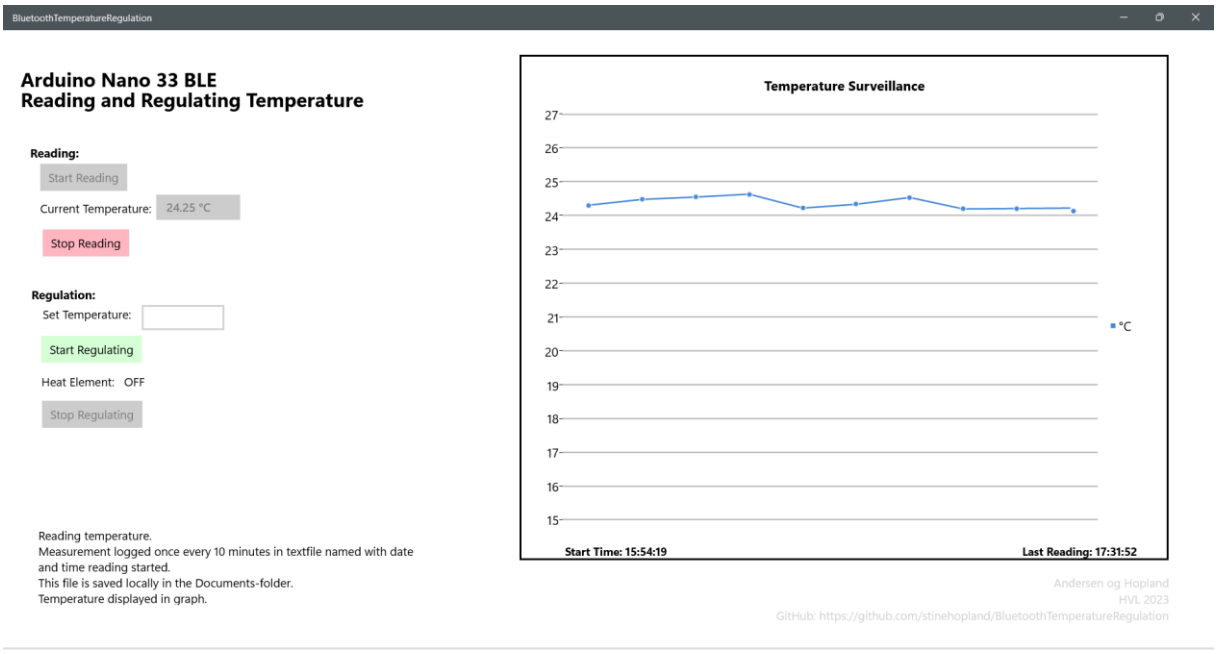


Figur 9: Skjermutklipp av ferdig brukergrensesnitt ved oppstart av programmet

Figur 9 viser et skjermutklipp av det ferdige brukergrensesnittet ved oppstart av programmet. Programmet består av fire forskjellige knapper, et display som viser nåværende temperatur, en tekstboks bruker kan skrive inn ønsket temperatur, en statusblokk som viser tilstanden til varmeelementet (ON/OFF), en statusblokk som viser aktuell informasjon til bruker og et display for graf med temperaturmålinger som dukker opp når man starter å lese av temperatur.

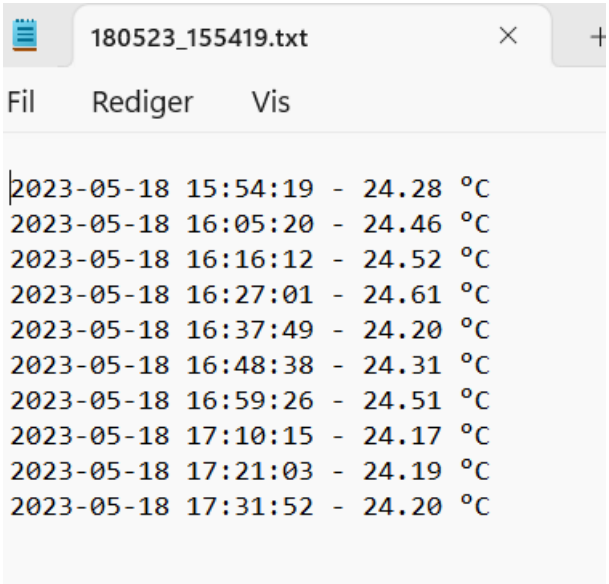
For å sikre at brukeren ikke kan gjøre noe i «feil rekkefølge» har vi sørget for at knappene for funksjoner som ikke er tilgjengelige, uten at man trykker på en annen knapp først, ikke er mulig å trykke på. Vi har delt opp avlesning og regulering slik at det er mulig for bruker å kun lese av temperatur uten å regulere, men det er ikke mulig regulere uten at man allerede leser av temperatur.

Statusblokken med informasjon til bruker oppdateres hver gang man trykker på en ny knapp. Hvilken informasjon som vises og hvilke prosesser som skjer i programkoden når man trykker på hver av knappene er beskrevet i kapittel 5.2.3.



Figur 10: Skjermbilde av brukergrensesnitt med temperaturovervåking

Figur 10 viser skjermbilde av hvordan programmet ser ut etter at vi har kjørt avlesing av temperatur i litt over én og en halv time. Som man ser i statusblokken nederst til venstre får bruker informasjon om at temperaturen leses, hvor ofte temperaturen skrives til tekstfil, hvor tekstfilen ligger og hva den heter. Grafen plotter avlest temperatur når avlesingen starter og plotter deretter temperatur hvert 10. minutt (samme som i tekstfil). Ettersom programmet skal kjøre over lang tid, og man kun trenger å se om temperaturen har holdt seg som den skal, fant vi ut at det enkleste og mest oversiktlige var å kun vise tidspunkt for første og siste måling i grafen. Skulle man skrevet ut tidspunkt for alle målingene langs x-aksen hadde tidspunktene lagt seg over hverandre etter hvert som grafen fylles opp.



Figur 11: Skjermbilde av tekstfil med lagrede målinger

Figur 11 viser et skjermbilde av tekstfilen som hører til kjøringen av program som vist i figur 10. Som vi ser blir hver temperaturmåling lagret med dato, tidspunkt og temperatur. Disse målingene er kjørt i romtemperatur uten regulering. Som vi også ser fra figur 11, er det en omtrentlig ett-minutts forsinkelse utover det ti-minutts intervallet det er ment å være mellom målingene. Dette skyldes at det er inkludert forsinkelse i flere funksjoner som arbeider sammen i løpet av den tiden som går mellom hver måling. Det er derimot ikke en stor problemstilling, ettersom intervallet strengt tatt ikke behøver å være nøyaktig 10 minutter, og intervallet kan også enkelt endres.

5.2.2 Unntakshåndtering

Unntakshåndtering er en sentral del av programkoden, ettersom det står for håndtering av potensielle feil og andre problemer ved bruk av applikasjonen. Vi har forsøkt å identifisere de mest sannsynlige feilene som kan oppstå, og implementert unntakshåndtering for å sikre at applikasjonen er så brukervennlig som mulig. Det første steget vi tok var å legge inn aktivering og deaktivering av knapper i koden, som nevnt i kapittel 5.2.1, for å hindre brukerfeil i form av at knappene blir trykket på «i feil rekkefølge». Dette gjør også at brukeren har færrest mulig «valg» og klarer å navigere mer intuitivt.

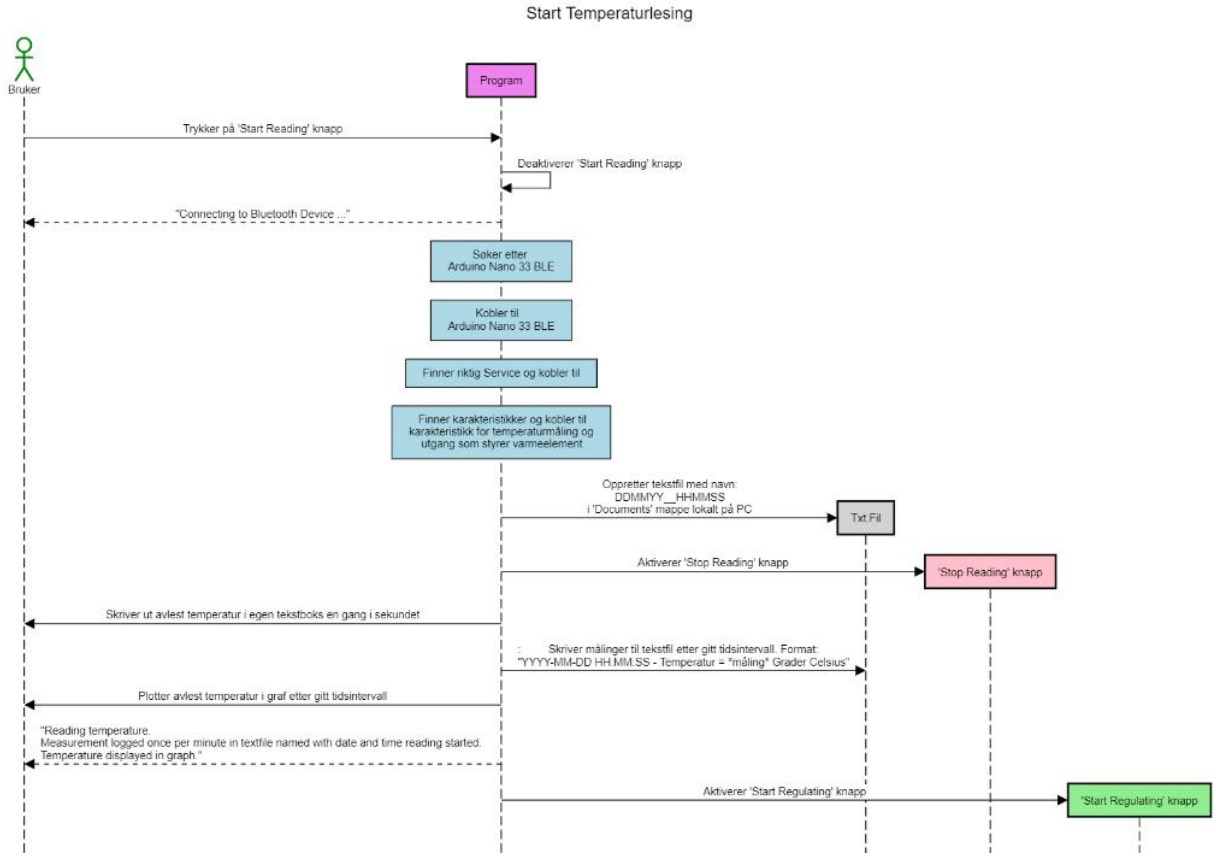
For at brukeren skal være trygg på at applikasjonen fungerer som den skal, har vi som nevnt tidligere lagt til en statusblokk som skriver ut meldinger om hva programmet gjør og andre instrukser. Dette fungerer også som unntakshåndtering, fordi det gir brukeren informasjon om hva vedkommende kan gjøre til enhver tid. Dersom brukeren gjør feil, har vi programmert inn feilmeldinger, som kommer opp som et pop-up vindu med den aktuelle feilmeldingen. Dette skjer eksempelvis dersom brukeren forsøker å skrive inn en ønsket temperaturverdi i feil format, altså med bokstaver eller andre tegn som ikke er tall.

5.2.3 Sekvensdiagram

For å få en oversikt over hvilke funksjoner programmet har og hva som skjer når man kjører programmet har vi laget sekvensdiagrammer for hver av de fire knappene i brukergrensesnittet.

Sekvensdiagrammene er laget via nettsiden: https://sequencediagram.org [13].

5.2.3.1 'Start Reading'



Figur 12: Sekvensdiagram; 'Start Reading' knapp

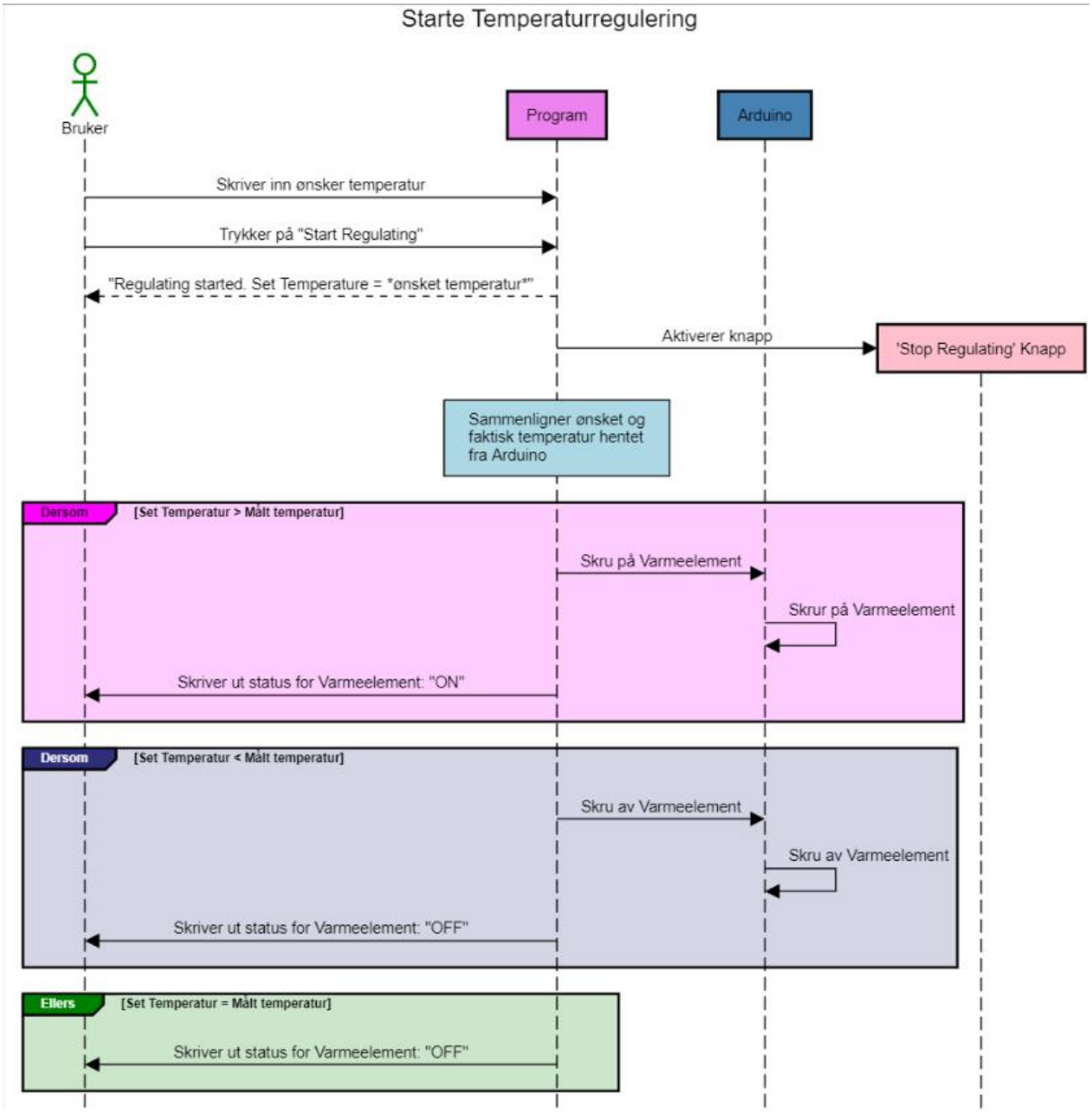
Figur 12 viser et sekvensdiagram over hva som skjer når man trykker på 'Start Reading' knappen i brukergrensesnittet. Det første programmet gjør er å deaktivere 'Start Reading' knappen slik at det ikke er mulig å trykke på denne flere ganger etter at man har begynt å koble til Arduino og allerede har startet å lese av temperatur. Programmet skriver ut beskjed til bruker om at den har startet prosessen med å koble seg til Arduino og søker etter Bluetooth-enhet med navnet vi har gitt i Arduino kode (Nano 33 BLE). Når enheten er funnet søker den etter servicen vi har definert i Arduino programmet og kobler seg til denne. Deretter finner den karakteristikkene for temperaturmålinger og styring av varmeelement som vi har definert i Arduino programkoden og kobler seg til disse.

Noe vi ikke har tatt med i sekvensdiagrammet for å spare plass, men som er en del av unntakshåndtering, er at dersom programmet ikke finner Arduino eller har problemer med å koble til service eller karakteristikker har vi lagt inn en funksjon som gjør at programmet slutter å søke etter 30 sekunder og avslutter alle prosesser i 'Start Reading' knappen. Bruker får beskjed om at kobling til Arduino feilet og at de må trykke på 'Start Reading' knappen for å prøve igjen. Dette har vi gjort for at programmet ikke skal lete i det uendelige, og at bruker kan sjekke at alt er i orden med Arduino før de eventuelt prøver å koble til igjen.

Dersom tilkobling til Arduino er suksessfull, oppretter programmet en tekstfil med filnavn på følgende form: 'DDMMYY_HHMMSS' (dato, måned, år, time, minutter, sekunder) som tilsvarer tidspunktet målingene startet. Filen blir lagret i 'Dokument' mappen lokalt på datamaskinen du kjører programmet på. Når dette er gjort aktiveres 'Stop Reading' knappen slik at bruker kan avslutte avlesning.

Programmet skriver ut avlest temperatur én gang i sekundet i eget display, og plotter første avlesning i grafen til høyre i brukergrensesnittet. Etter dette plottes temperaturmålingene etter et gitt tidsintervall (hvert 10. minutt slik det er innstilt da vi leverer programmet). Bruker får beskjed om at avlesningen har startet, at målingene blir lagret i tekstfil med gitt format og at temperaturmålingene blir vist i grafen. Til slutt aktiverer programmet 'Start Regulating' knappen slik at bruker kan sette ønsket temperatur og starte regulering.

5.2.3.2 'Start Regulating'



Figur 13: Sekvensdiagram; 'Start Regulating' knapp

Figur 13 viser sekvensdiagram for hva som skjer i programmet når man vil starte reguleringen av temperatur. Det første brukeren må gjøre er å skrive inn ønsket temperatur i gitt tekstfelt og trykke på knappen 'Start Regulating'. Brukeren får beskjed om at temperaturreguleringen har startet og hva temperaturen er stilt inn til. Programmet aktiverer 'Stop Regulating' knappen, og begynner å sammenligne temperaturmålingene fra temperatursensor og ønsket temperatur gitt av brukeren.

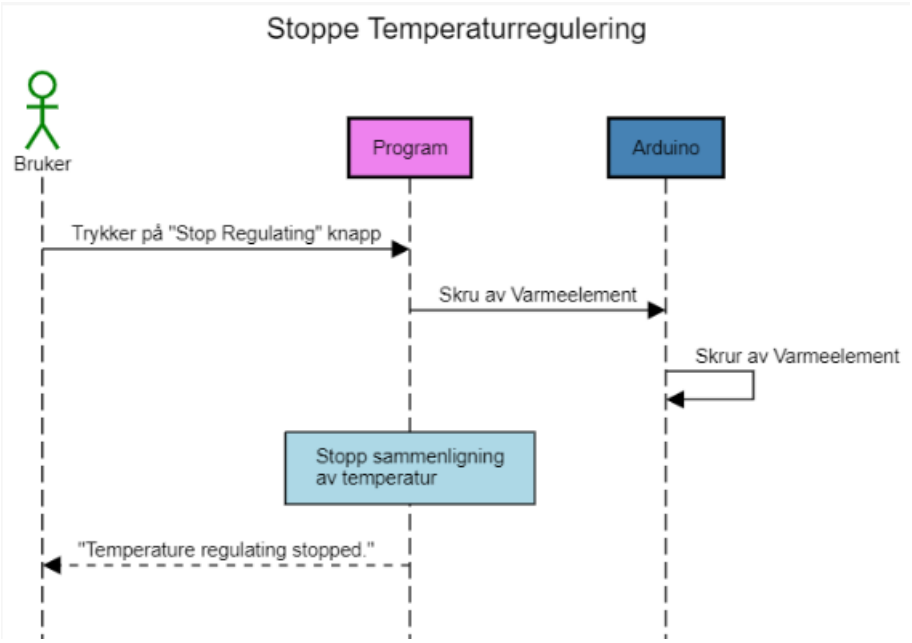
Dersom ønsket temperatur er høyere enn avlest temperatur endrer programmet Bluetooth-karakteristikken som styrer varmeelementet via Arduino til å skru på varmeelementet. Arduino programmet leser av denne karakteristikken én gang i sekundet, så når denne leses av som 'høy' setter

den utgangen til releet som styrer varmeelementet på. Statusen for varmeelementet i brukergrensesnittet endres til «ON».

Dersom ønsket temperatur er lavere enn målt temperatur endres karakteristikken til varmeelementet til å skru av varmeelementet, Arduino skrur av utgangen som styrer varmeelementet og status for varmeelementet viser «OFF» i brukergrensesnittet.

Dersom ønsket verdi er verken høyere eller lavere enn målt temperaturverdi skrives status i brukergrensesnitt «OFF».

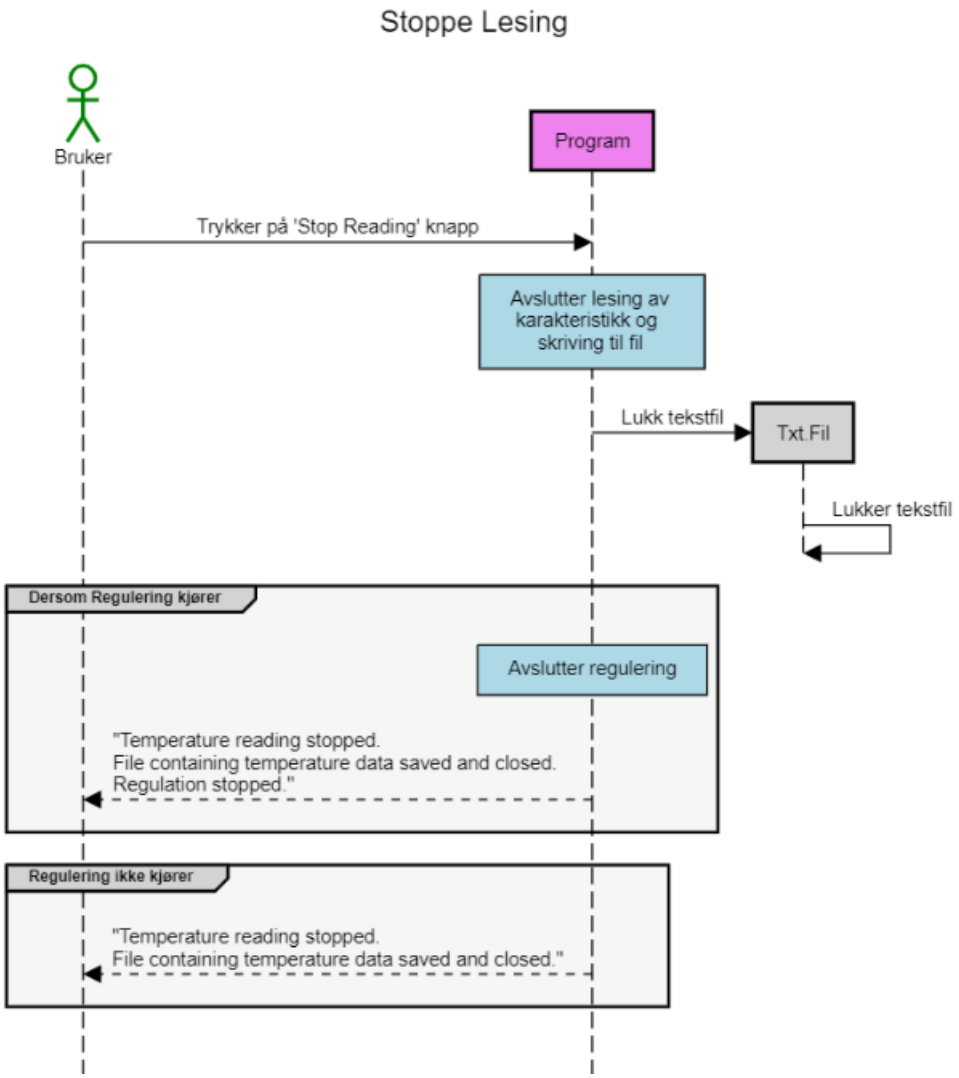
5.2.3.3 'Stop Regulating'



Figur 14: Sekvensdiagram; 'Stop Regulating' knapp

Figur 14 viser hva som skjer i programmet når bruker trykker på knappen 'Stop Regulating' og avslutter reguleringen av temperatur. Det første som skjer er at programmet endrer karakteristikken som hører til varmeelementet, slik at Arduino skrur av utgang som hører til releet og skrur av varmeelementet. Deretter avslutter programmet sammenligning av temperatur og bruker får beskjed om at temperaturreguleringen er avsluttet. Programmet vil fortsatt lese av temperatur og plotte avlesninger etter gitt tidsintervall i graf.

5.2.3.4 'Stop Reading'



Figur 15: Sekvensdiagram; 'Stop Reading' knapp

Figur 15 viser sekvensdiagram for hva som skjer i programmet når man trykker på knappen 'Stop Reading' for å avslutte lesing av temperatur. Det første programmet gjør er å avslutte lesing av temperatur karakteristikken og skriving til tekstfil. Tekstfilen lukkes. Dersom regulering kjører vil programmet avslutte reguleringen og gi beskjed til bruker om at avlesning av temperatur er avsluttet, fil med temperaturdata er lagret og lukket og at reguleringen har sluttet. Dersom programmet ikke regulerer temperatur når brukeren stopper avlesningen vil brukeren få de samme beskjedene med unntak av beskjed om at regulering har stoppet.

6 Testing av programvare

Ved utvikling av programvare vil man alltid teste programmet underveis etter hvert som nye funksjoner blir implementert, for å sikre at alt fungerer slik som det skal før man går videre. Ettersom utvikling av UWP applikasjonen utvilsomt var den delen av prosjektet som har tatt lengst tid, har vi måttet teste mange ulike funksjonaliteter i løpet av den tiden vi har jobbet med programmet. Disse testene har derimot vært veldig uformelle, og vi vil kun ta for oss testing av funksjonene som hører til det ferdige brukergrensesnittet i dette kapittelet.

6.1 Brukertesting og evaluering av brukergrensesnitt

6.1.1 Funksjonell testing av brukergrensesnitt

Funksjonell testing av brukergrensesnittet er viktig for å sikre at programmet fungerer slik som forventet. Vi ønsker å teste hver funksjon i applikasjonen individuelt, for å deretter teste hele programmet som et komplett system. Tabellen under viser hva programmet skal gjøre når brukeren trykker på de ulike knappene i brukergrensesnittet (dette er også beskrevet i sekvensdiagrammene i kapittel 5.2.3). Testene anses som vellykket dersom handlingene blir utført slik som beskrevet i denne tabellen.

Knapp	**Handling ved vellykket test**
Start Reading	Leter etter enhet med navnet "Nano 33 BLE", og etablerer en tilkobling dersom enheten oppdages. Leter etter og kobler til de egendefinerte servicen og de tilhørende karakteristikkene. Oppretter en txt-fil hvor temperaturmålingene blir lagret. Skriver ut melding til brukeren om at programmet leser temperaturverdier og logger målingene hvert 10 minutt. Målingene vises i grafen med tidspunkt for start av overvåkning og seneste avlesning.
Start Regulating	Temperaturverdien oppgitt av brukeren blir lagret som settverdi. Skriver ut melding til brukeren om at temperaturreguleringen er iverksatt med ønsket settverdi. Settverdien sammenlignes kontinuerlig med de seneste temperaturmålingene. Dersom temperaturen er 1 grad lavere enn settverdien, skriver programmet en byte-array med verdi 0x01 til Arduino og slår varmeelementet på. Dersom temperaturen har nådd settverdien, skrives 0x00 til Arduino og varmeelementet slås av.

Stop Regulating	Stopper reguleringen ved å gi beskjed til Arduino om å slå av varmeelementet og stoppe sammenligningen av temperaturmålingene mot settverdien. Skriver ut melding til brukeren om at reguleringen er stoppet.
Stop Reading	Fjerner innholdet i tekstboksen som viser seneste temperaturmåling. Avbryter loopen som står for lesing av temperaturdata. Lukker og lagrer tekstfilen som inneholder temperaturmålingene. Sjekker om det foregår en reguleringsprosess, og dersom det gjør det blir den stoppet av programmet. Skriver ut en melding til brukeren om at temperatur avlesing er stoppet og at filen som inneholder målingene er lukket og lagret. Dersom en reguleringsprosess ble stoppet, opplyses dette også om i meldingen til brukeren.

Vi ønsket også å teste brukeropplevelsen av applikasjonen, og da spesielt at brukergrensesnittet oppleves som enkelt, at statusmeldinger og feilmeldinger blir skrevet ut for brukeren, samt at programmet i sin helhet gjør det det skal. Dette ble gjort ved at en medstudent som ikke hadde noen forhåndskunnskaper om applikasjonen ble spurt om å teste det og gi tilbakemeldinger. En vellykket test av brukeropplevelsen ville ved dette tilfellet vært at medstudenten rapporterte en god brukeropplevelse samt at vedkommende forsto essensen av hvordan programmet fungerte.

6.1.1.1 Resultat av funksjonell testing

Testing av 'Start Reading'-knapp var en suksess, som verifiserer at UWP applikasjonen kan etablere en tilkobling til Arduino, samt lese temperaturen fra SHT31 sensoren. Det ble også bekreftet at programmet lykkes med å åpne og skrive til en tekstfil som ble opprettet på valgt filplassering. Til slutt verifiserte denne testen at tekstboksen som viste målingene hvert sekund, i tillegg til grafen som ble oppdatert hvert 10 minutt, fungerte slik de skulle. Testen av 'Start Regulating'-knappen var også en suksess, og den bekreftet at UWP applikasjonen klarte å regulere temperaturen. Disse resultatene forklares ytterligere i kapittel 6.2. Testene av 'Stop Regulating' og 'Stop Reading' gav også suksess, da henholdsvis reguleringen og avlesing stoppet slik som det skulle. Tekstfilen som inneholdt alle målingene ble også lukket og lagret lokalt på datamaskinen.

Testingen av brukeropplevelsen ble også ansett som en suksess. Det ble ikke trigget noen feilmeldinger eller unntakstilstander da vår medstudent testet programmet og gjorde det han intuitivt tenkte han måtte gjøre da han skulle kjøre målinger og regulering. På et tidspunkt ba vi ham om å forsøke å trigge en feilmelding for å se om det dukket opp noe som ikke burde skje. Den eneste feilmeldingen som ble trigget da, var den som kommer opp når brukeren skriver et ugyldig tall som settverdi og viser at denne funksjonen fungerer som den skal. Samlet sett fikk vi tilbakemeldinger om en god brukeropplevelse og

at applikasjonen var lett å forstå. Det var spesielt én funksjon som viste seg å gi god brukervennlighet, og det var den som aktiverer og deaktiverer knappene på brukergrensesnittet avhengig av hva bruken skal tillates å gjøre til enhver tid. Det er eksempelvis ikke mulig å trykke på noe annet enn 'Start Reading' når programmet først åpnes, ettersom denne må trykkes på først. Dette tror vi bidro i stor grad til at medstudenten som testet applikasjonen ikke trigget noen unntakshendelser som kunne føre til programsvikt.

Som konklusjon ser vi at programmet fungerer som det skal, det er enkelt å bruke og sikkert når det kommer til rekkefølge av prosesser og at feilmeldinger dukker opp når programmet får ugyldig input. Vi fikk likevel noen tilbakemeldinger på designet av programmet som ville gjøre det hakket mer brukervennlig, og under har vi listet endringene som ble gjort for å rette opp i dette:

- Statusmeldingene ble endret fra grå tekst til svart tekst for bedre synlighet.
- Knappen som initierer reguleringen endret navn fra 'Set Temperature' til 'Start Regulating'.
- 'Stop Reading'-knappen ble ikke deaktivert etter den ble trykket på, men det er nå fikset.

6.1.2 Mangler i programmet

Til tross for god brukeropplevelse ved testing av brukergrensesnittet har vi selv oppdaget flere mangler ved programmet. Den største mangelen vi ser er at det bør være enda bedre unntakshåndtering, både når det kommer til brukerfeil, men også når det kommer til uforutsette feil eller brudd i systemet. I dette kapittelet har vi listet noen mangler og funksjoner som vi ønsker å fikse, men ikke har hatt tid til før prosjektslutt. Disse bør implementeres for å bedre brukervennligheten og funksjonaliteten til programmet, samt hindre at programmet kræsjer.

Set-Temperatur med desimaler

Dersom man skriver komma fremfor punktum når man skal skrive ett desimaltall som ønsket temperatur, vil programmet ikke forstå at det skal være ett komma der og skrive ut temperaturen uten desimalskille. Det vil si at dersom du for eksempel skriver inn 30,5 som ønsket temperatur vil programmet lese det som ett heltall og sette ønsket temperatur som 305 °C fremfor 30,5 °C. Dersom man skriver 30.5 derimot vil programmet skrive ut riktig temperatur.

Unntakshåndtering dersom Bluetooth tilkobling detter ut

Dersom Bluetooth tilkoblingen faller ut eller Arduino mister strømforsyning mens programmet leser av temperaturen, vil programmet avsluttes av seg selv uten forvarsel. Tekstfilen med målinger vil være lagret med målingene frem til koblingen forsvant. Det bør altså legges inn en unntakshåndtering dersom dette skjer slik at bruker får opp en feilmelding og/eller at programmet prøver å koble seg til Arduino igjen automatisk, istedenfor at programmet avsluttes.

Sjekk av tilkoblede komponenter

Det er ikke implementert noe sjekk av sensor, relé eller varmeelement. Dersom noen av de tilkoblede elementene ikke fungerer må dette oppdages av brukeren.

6.2 Testing av regulering

Dette kapittelet tar for seg testing av temperaturregulering gjennom brukergrensesnitt. Vi vil ta for oss fremgangsmåte for gjennomføring av tester, resultater av testene og til slutt diskutere resultatene.

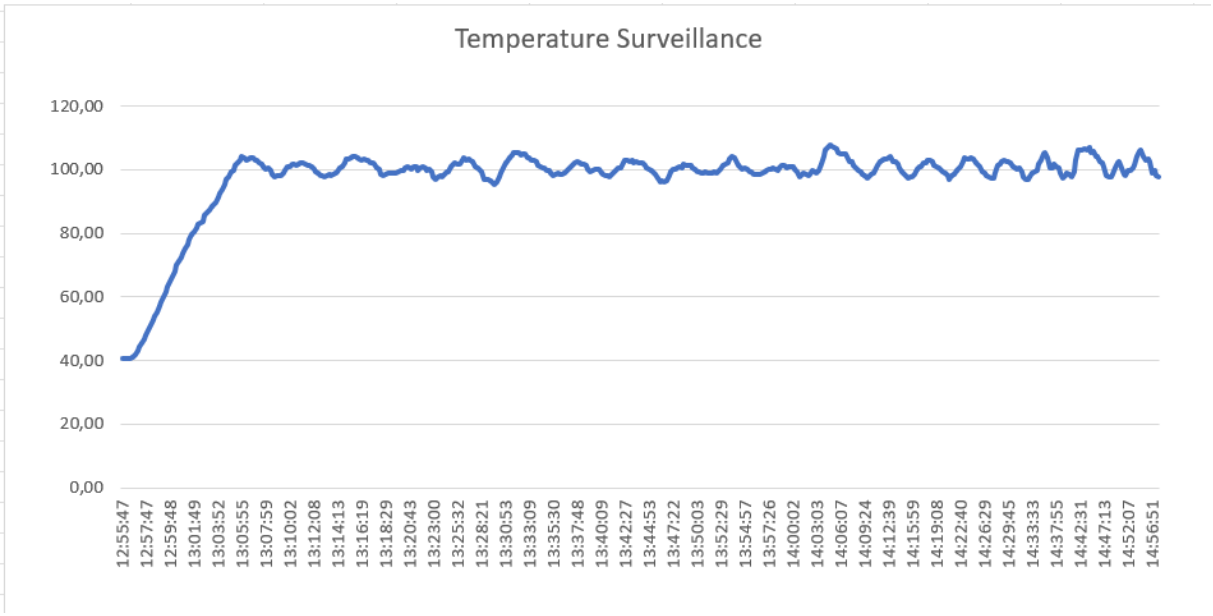
6.2.1 Gjennomføring

Målet med testingen er å se om programmet skrur av og på varmeelementet når den skal, om temperaturen holder seg i nærheten av ønsket verdi og at den trådløse koblingen holder seg stabil når den kjøres over tid.

Dersom programmet skrur av og på temperaturen når den beveger seg utenfor +/- 1 °C av valgt sett-verdi vil det fungere bra nok i forhold til hvordan systemet skal brukes på FARLAB. Vi gjorde derfor veldig enkel testing ved å feste selve temperaturmåleren rett på varmeelementet i kretsen. Når dette systemet skal brukes på FARLAB, vil sensoren måle lufttemperatur i en lukket beholder. Det er derfor ikke optimal testing og vi vet på forhånd at vi ikke vil klare å holde temperaturen stabil på denne måten. Likevel er både vi og oppdragsgiver enig om at denne testingen er god nok for å teste at systemet skrur av og på varmeelementet når den skal, og at Bluetooth tilkoblingen fungerer som den skal. Måleoppsettet er vist i figur 43 i Appendiks E.4.

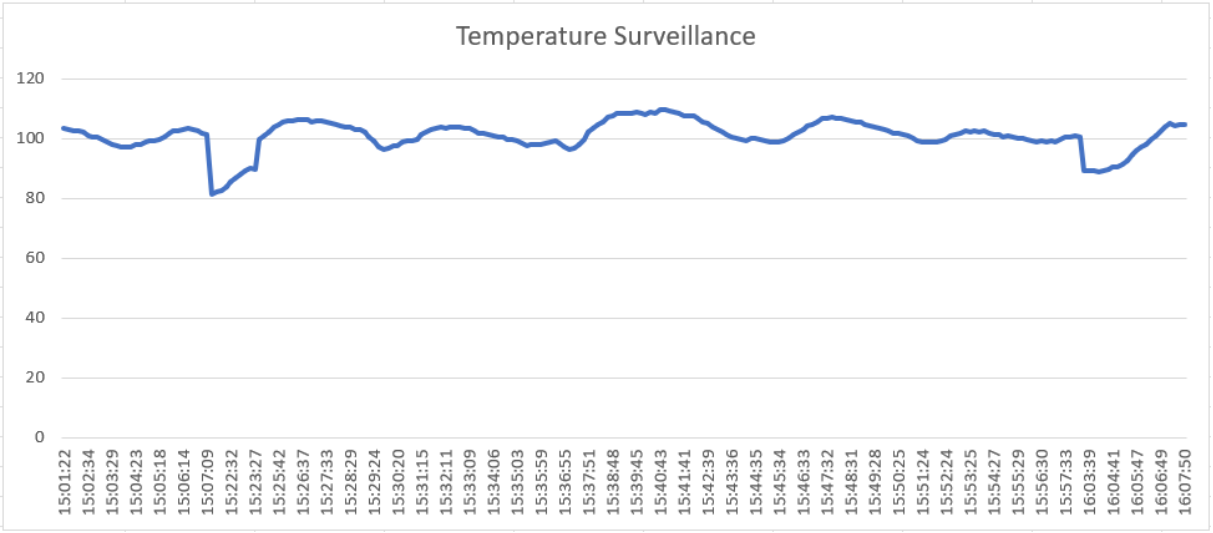
6.2.2 Resultater

For å teste reguleringen over tid satte vi ønsket temperatur til 100 °C og lot programmet kjøre i 1 og 2 timer i strekk. Vi satt tidsintervallet for lagring av data og plotting i graf til hvert 10. sekund for å kunne følge nøye med på utviklingen. Resultatet av testene er vist i figur 16 og 17 på neste side.



Figur 16: Regulert temperatur fra testing over 2 timer plottet i Excel fra txt-fil (24.04.2023)

Figur 16 viser den første testen vi kjørte over tid. Denne testen gikk over 2 timer. Starttemperaturen på varmeelementet var allerede 40 °C da vi startet, og vi satt ønsket temperatur til 100 °C. Som vi kan lese av grafen tok det noen minutter før ønsket temperatur var nådd. Vi kunne se i brukergrensesnittet at varmeelementet ble skrudd av da temperaturmålingen viste 100 °C, men ettersom varmeelementet fortsatt holder varmen en stund etter at det blir skrudd av får vi litt oversving før temperaturen begynner å synke igjen. Idet temperaturen når 99 °C skrus varmeelementet på igjen, og det tar ikke lange tiden før den begynner å stige igjen. Høyeste målte verdi: 107,72 °C. Laveste målte verdi etter at Set-temperatur er nådd: 95,44 °C.



Figur 17: Regulert temperatur fra testing over 1 time plottet i Excel fra txt-fil (24.04.2023)

Vi kjørte en ny test på litt over én time med samme tidsintervall for lagring av temperaturdata. Resultatene av denne testen ser du i figur 17. Her var temperaturen allerede over 100 °C da vi startet testen.

Høyeste målte temperatur: 109,81. Laveste: 81,18.

Vi hadde to store dropp i temperatur mens vi kjørte reguleringen. Vi trodde først dette hadde noe med gjennomtrekk og luften rundt sensoren å gjøre, men da vi gikk gjennom tekstfilen for målingene fant vi ut at det var perioder der måling ikke var registrert.

Dato	Tidspunkt	Temperatur
24.04.2023	15:07:09	101.37 °C
24.04.2023	15:21:48	81.18 °C
24.04.2023	15:58:11	100.49 °C
24.04.2023	16:03:27	89.43 °C

I tabellen over kan man se målingene fra de aktuelle tidspunktene. I første dropp gikk det over 14 minutter mellom avlesningene, og i det andre droppet gikk det litt over 5 minutter.

Grunnen til at målingene stoppet og begynte igjen på senere tidspunkt viste seg å være at programmet stopper å kjøre dersom vi minimerer applikasjonsvinduet. Vi fant ut at Universal Windows Platform applikasjonene har en innebygd funksjon som gjør at programmet pauser kjøring når programmet minimeres for å redusere bruk av prosessorkraft [14]. Dette har vi klart å fikse, og applikasjonen fortsetter nå å lese av temperatur og regulere den selv om man minimerer applikasjonsvinduet.

Resultatet av testene viser at systemet fungerer som det skal, og vi fikk avdekket og fikset et stort problem med programmet.

7 Diskusjon

I dette kapittelet tar vi for oss selve prosessen med prosjektarbeidet – altså hvordan det har gått i forhold til hva vi så for oss når vi planla arbeidet.

I forkant av prosjektet gjorde vi en forstudie, som blant annet besto av å utarbeide en fremdriftsplan (se Appendiks B.3) og risikoanalyse (se Appendiks A.3) for arbeidet vi skulle i gang med. Da vi satt opp fremdriftsplanen valgte vi å sette oss delmål som ikke var alt for spesifikke, fordi vi var usikre på hvor lang tid vi kunne forvente å bruke på de ulike delene av prosjektarbeidet. Vi antok at programmeringen kom til å være mest tidkrevende, og dette viste seg å være helt korrekt. Vi planla også at det ville ta flere uker å implementere BLE kommunikasjon i programkoden siden vi ikke

hadde noen forkunnskaper på dette området, noe som også stemte godt overens med realiteten. Alt i alt har fremdriften holdt seg godt innenfor rammene vi satt opp i den opprinnelige fremdriftsplanen, men vi har måtte gjøre et par justeringer. Vi begynte ikke på rapportskrivingen før vi var helt ferdig med programmeringen, og selve programmeringen tok en uke lenger enn det som var planlagt på forhånd. Dette resulterte i at vi ikke begynte å skrive før i uke 17, selv om vi egentlig hadde planlagt å starte med skrivingen allerede i uke 14. Utenom dette har vi klart å holde oss innenfor den generelle tidsrammen, og holdt den progresjonen vi håpet vi skulle klare ellers i prosjektet.

Angående risikoanalysen har vi støtt på svært få uforutsette problemer underveis. Risikoanalysen handlet om å vurdere risikoen for diverse hendelser som kunne ha innvirkning på fremdriften av prosjektet, og i løpet av arbeidet har ingen av problemstillingene vært særlig aktuell. Heldigvis for oss har det vært lite sykdom blant gruppemedlemmene, og det har ikke oppstått noen komplikasjoner når det gjelder risiko bundet til arbeidet ved FARLAB eller andre risikoelementer forbundet med miljøet rundt oppgaven. Bluetooth-tilkobling har vært det eneste som har påvirket prosjektet i praksis, men problemet har i størst grad vært i tilknytning til utviklingen av UWP applikasjonen.

Den største utfordringen vi har hatt i løpet av dette prosjektet, med god margin, har vært det å programmere med Universal Windows Platform. Ingen av oss hadde tidligere erfaring med dette, og det tok en del tid å sette seg inn i programmet og gjøre seg kjent med hvordan det fungerer i forhold til Windows Forms som vi var kjent med og har brukt tidligere. Den utvilsomt største utfordringen med programmeringen var at i UWP må man skrive XAML kode til å utforme designet av brukergrensesnittet. Dette viste seg å være ganske krevende, men vi har lært veldig mye og er fornøyd med hva vi har fått til rundt design og layout selv om det fremdeles er rom for forbedringer (se kapittel 8.1.6).

8 Fremtidig arbeid

Som nevnt i kapittel 6.1.2., er det noen mangler i programmet som bør forbedres og videreutvikles. I tillegg til disse vil dette kapittelet ta for seg forslag til fremtidig arbeid som ikke er essensielt, men som kan legges til for å ytterligere øke funksjonaliteten til programmet.

8.1.1 Endring av intervall for lagring til tekstfil

Slik programmet er nå må man inn i selve programkoden og endre programmet for å endre intervallet for lagring til tekstfil og plotting i graf. For å slippe å gjøre dette hver gang hadde det vært ideelt om bruker hadde mulighet til å legge inn intervall selv. Da ville vi gjerne hatt et forhåndsdefinert standardintervall som bruker kan endre dersom de ønsker det.

8.1.2 Automatisk tilpassing av punktet i graf

Slik programmet er nå plottes det punkter i grafen etter gitt tidsintervall. Når dette står til for eksempel 10 minutter, som det gjerne vil gjøre når vi har temperaturregulering over et døgn om gangen, så vil det ta veldig lang tid før man ser det andre punktet i grafen. For å kunne følge med på utviklingen i temperatur fra programmet starter avlesing hadde det vært bedre om grafen skrev ut flere punkter i starten av avlesningen og heller slettet gamle og overflødige punkter når grafen jevner seg ut.

8.1.3 Velge tekstfil og filplassering

Slik programmet er nå vil det opprettes en ny tekstfil hver gang man starter avlesning av temperatur. Dersom man stopper avlesningen vil det opprettes en ny tekstfil når man starter avlesning igjen. For å kunne pause avlesningen og fortsette i samme fil burde det opprettes en mulighet for å velge tekstfil for lagring dersom man skulle ønske det, men fortsatt ha oppretting av ny tekstfil som 'default' setting. Det kan også være greit for bruker å kunne velge filplassering via brukergrensesnittet.

8.1.4 Mulighet til å koble fra/til Arduino på nytt

Slik programmet er nå kobler programmet seg til Arduino når man starter lesingen første gang og holder koblingen helt til man avslutter programmet. Dersom man skulle ville koble fra Arduino av en eller annen grunn mens programmet kjører må dette legges inn.

8.1.5 Velge mellom grader Celsius og grader Fahrenheit

Mulighet for bruker å kunne velge om temperatur skal leses av i grader Celsius eller grader Fahrenheit.

8.1.6 Gjøre programmet kompatibel med alle skjermstørrelser

Programmet ble designet for å passe 15 tommers skjerm. Det er ikke optimalisert for å kunne tilpasses alle formater av skjermer, så dette kan utbedres.

9 Konklusjon

I starten av dette prosjektet brukte vi en vesentlig mengde tid på å få til BLE kommunikasjonen mellom UWP applikasjonen og Arduino. Etter at dette kom på plass, har programmeringen stått i fokus og målet har vært å lage en god og brukervennlig applikasjon. Det er uten tvil Bluetooth komponenten og programmering som har vært mest tidkrevende, men vi kom heldigvis i mål tidsnok til å fullføre prosjektet.

Det som var viktig i denne oppgaven var å få laget et program som kunne brukes til å regulere og overvåke temperatur. Dette har vi fått til. Reguleringen er ikke veldig nøyaktig slik den er nå, men programkoden vår kan relativt enkelt modifiseres og dermed er det mulig å gjøre reguleringen mer presis om man ønsket dette i fremtiden. Kommunikasjonen via Bluetooth fungerer som den skal, og

programmet oppfyller oppdragsgivers krav og instruksjoner. Vi ønsker å gjøre noen enkle forbedringer når det kommer til unntakshåndtering, men er ellers svært fornøyd med det ferdige resultatet.

Oppdragsgiver har understreket at programvaren var den viktigste delen av prosjektet vårt, ettersom en del av de fysiske komponentene sannsynligvis ville bli byttet ut i fremtiden. Dette har vi tatt høyde for, både i utviklingen av Arduino programmet og UWP applikasjonen. Det er høyst aktuelt å erstatte temperatursensoren for å gjøre systemet mer kompatibelt med temperaturene som må håndteres på FARLAB.

Det ferdige resultatet av prosjektet er noe vi samlet sett er veldig stolt av, og vi håper også at oppdragsgiver er fornøyde med både applikasjonen og systemet i sin helhet. Forhåpentligvis kan programmet vårt hjelpe med deres forskning, og være med på å løse de utfordringene som fulgte med deres nåværende system for overvåking- og regulering av temperatur. Vi er også engasjert og villig til å bidra med å løse eventuelle utfordringer som skulle oppstå med systemet etter endt prosjekt.

10 Referanser

[1] Universitetet i Bergen. «Fakta tall ved GEO» uib.no. Hentet fra: https://www.uib.no/geo/55218/fakta-tall-ved-geo (Lastet ned: 17.01.2023).

[2] Universitetet i Bergen. «FARLAB,» uib.no. Hentet fra: https://www.uib.no/en/FARLAB (Lastet ned: 26.01.2023).

[3] Universitetet i Bergen. «Mat253 and Kiel IV,» uib.no. Hentet fra: https://www.uib.no/en/FARLAB/119770/mat253-and-kiel-iv (Lastet ned: 19.05.2023).

[4] Universitetet i Bergen, «Mat253Plus and Kiel IV,» Hentet fra: https://www.uib.no/en/FARLAB/119771/mat253plus-and-kiel-iv (Lastet ned: 19.05.2023).

[5] Universitetet i Bergen, «Clumped isotopes in carbonates and CO2,» uib.no. Hentet fra: https://www.uib.no/en/FARLAB/119787/clumped-isotopes-carbonates-and-co2 (Lastet ned: 19.05.2023).

[6] J. Huang. "Grove – Temp and Humi Sensor(SHT31)" seeedstudio.com. Hentet fra: https://wiki.seeedstudio.com/Grove-TempAndHumi_Sensor-SHT31/ (Lastet ned: 02.02.2023).

[7] Peak Sensors. «What is a Pt100 sensor?" peaksensors.com. Hentet fra: https://peaksensors.co.uk/what-is/pt100-sensor/ (Lastet ned: 21.05.2023).

[8] J. Bagur, T. Chung, K. Söderby. «Powering Alternatives for Arduino Boards» docs.arduino.cc. Hentet fra: https://docs.arduino.cc/learn/electronics/power-pins (Lastet ned: 02.02.2023).

[9] Bluetooth SIG, «The Bluetooth Low Energy Primer,» bluetooth.com. Hentet fra: https://www.bluetooth.com/bluetooth-resources/the-bluetooth-low-energy-primer/?utm_source=internal&utm_medium=blog&utm_campaign=technical&utm_content=the-bluetooth-low-energy-primer (Lastet ned: 27.04.2023).

[10] Bluetooth. «Bluetooth Wireless Technology» Bluetooth. Hentet fra: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/ (Lastet ned: 12 mai 2023).

[11] Bluetooth SIG. «Assigned Numbers,» (16.05.2023). Utgitt av Bluetooth. https://btprodspecificationrefs.blob.core.windows.net/assigned-numbers/Assigned%20Number%20Types/Assigned_Numbers.pdf

[12] Microsoft. «Bluetooth Low Energy Sample,» Hentet fra: https://learn.microsoft.com/en-us/samples/microsoft/windows-universal-samples/bluetoothle/ (Lastet ned: 27.04.2023).

[13] «UML Sequence Diagram Online Tool,» 2014. Hentet fra: https://sequencediagram.org

[14] A. Ashcraft, M. Wojciakowski. "Run in the background indefinitely" microsoft.com. Hentet fra: https://learn.microsoft.com/en-us/windows/uwp/launch-resume/run-in-the-background-indefinitely (Lastet ned: 19.05.2023).

[15] A. Ashcraft, M. Wojciakowski. «Postpone app suspension with extended execution,» microsoft.com. Hentet fra: https://learn.microsoft.com/en-us/windows/uwp/launch-

resume/run-minimized-with-extended-execution#run-while-minimized
(Lastet ned: 21.05.2023).

[16] Arduino Nano 33 BLE datablad: "Product Reference Manual (SKU: ABX00030)." (05 mai, 2023). Arduino. https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf

[17] Arduino. «Pinout Diagram» arduino.cc. Hentet fra: https://store.arduino.cc/products/arduino-nano-33-ble?_gl=1%2A1if0gll%2A_ga%2AMTIxMjMzMDQ2NS4xNjc5OTk2MDUx%2A_ga_NEXN8H46L5%2AMTY4Mzk3OTE4NC42LjAuMTY4Mzk3OTE4NC4wLjAuMA… (Lastet ned: 13.05.2023).

[18] Datasheet SHT3x-DIS: Sensirion: The sensor company, Des. 2022. https://sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf

[19] Schneider Electric. "SE Relays" se.com. Hentet fra: https://www.se.com/us/en/product/6425AXXSZS-DC3/relay-se-relays-solid-state-spst-no-25a-48530-vac-3-32-vdc-uc-scr-ac-zero-cross-screw-terminal/ (Lastet ned: 24.04.2023).

[20] Store Norske Leksikon. «Dielektrikum» snl.no. Hentet fra: https://snl.no/dielektrikum (Lastet ned: 16.05.2023).

[21] Omega. «What is a cartridge heater and how does it work?» omega.com. Hentet fra: https://www.omega.com/en-us/resources/cartridge-heaters (Lastet ned: 16.05.2023).

[22] UUID Generator. «Online UUID Generator". Hentet fra: https://www.uuidgenerator.net/

[23] nRF52840 datablad: «nRF52840 Product Specification (v1.1)" Nordic Semiconductor, Feb, 2019. https://content.arduino.cc/assets/Nano_BLE_MCU-nRF52840_PS_v1.1.pdf

[24] Arduino Docs. "Arduino Nano 33 BLE" docs.arduino.cc. Hentet fra: https://docs.arduino.cc/hardware/nano-33-ble (Lastet ned: 22.05.2023).

Appendiks A Lister

A.1 Forkortelser og ordforklaringer

AC	Vekselstrøm
Arduino	Mikrokontroller
Arduino IDE	Program for å programmere mikrokontrolleren
BLE	Bluetooth Low Energy
C#	Programmeringsspråk
ELE102	Emnekode HVL: Programmering & Mikrokontrollere
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
I/O	Input/Output
I2C	Inter-Integrated Circuit
Mesh	Mange-til-mange kommunikasjon
PID	Proporsjonal – Integrering – Derivering
SCL	Seriell klokke
SDA	Seriell data
Solid state relé	Elektronisk bryter som skrus av og på når den mottar ekstern spenning
TXT	Tekstfil type
UWP	Universal Windows Platform
V	Volt, enhet for spenning
Visual Studio	IDE brukt for å programmere prosjekter i C#
W	Watt, enhet for effekt
Windows Forms	Verktøy for å utvikle GUI
XAML	Extensible Application Markup Language

A.2 Figurliste

A.3 Risikoliste

Risiko	Sannsynlighet	Alvor	Konsekvens	Tiltak
Sykdom	Middels	Lav	Midlertidig hindring i fremdrift	Fordele arbeidsmengde, arbeide over nett, digitale møter
Utfordringer med Bluetooth tilkobling	Middels	Middels	Arduino mister kommunikasjon med PC – temperatur satt av bruker blir ikke registrert	Skrive inn kode for sammenligning av målt temperatur og satt temperatur i Arduino program slik at det fortsetter å kjøre
Arbeid rundt gass under trykk	Lav	Høy	Helseskader, ekstreme tilfeller: død	Sikkerhetskurs, følge med på eventuelle alarmer
Ødelegge prøver på grunn av feil på temperaturstyring	Middels	Middels	Prøver må kastes. Mye penger går til spille	Omfattende testing før systemet kobles til instrumenter

Appendiks C Materialer

Liste over materialer	Formål
Arduino Nano 33 BLE	Leser av temperatur fra temperatursensor og sender videre til datamaskin. Skrur av og på varmeelement
Grove SHT31	Temperatursensor brukt til testing av system
Micro USB kabel	Brukes til strømforsyning og testing av Arduino program før kobling via Bluetooth
USB strømadapter	Forsyner Arduino med strøm
Koblingsbrett	Brett Arduino festes på for å enkelt kunne koble til temperatursensor og relé på riktige pinner
Koblingskabler	Brukes for å koble sammen Arduino, temperatursensor og relé
Solid state relé	Brukes til å skru av og på varmeelement
Varmeelement	Skrus av og på for å regulere temperatur
PC	PC med brukergrensesnitt for å kontrollere og overvåke temperatur

Appendiks B Prosjektledelse og styring

Prosjektstart var i januar 2023 etter første møte med oppdragsgiver. På dette møte ble det avklart en del knyttet til praktisk gjennomføring av prosjektet, slik som arbeidssted og faste møtetidspunkter. Mye av arbeidet ble gjort ved Realfagsbygget på UiB, ettersom vår oppdragsgiver har sin arbeidsplass her og det var lettere å kommunisere jevnlig og enkelt få kontakt dersom det skulle oppstå behov for oppklaring rundt oppgaven. Øvrig arbeid har blitt gjennomført på HVL Kronstad eller individuelt hjemme. Møte med intern veileder har blitt gjennomført fast annenhver uke, med mulighet for flere møter når prosjektets innleveringsdato nærmet seg. Det er blitt skrevet referat til hvert av møtene og tema før møte avklart på forhånd.

B.1 Prosjektorganisasjon

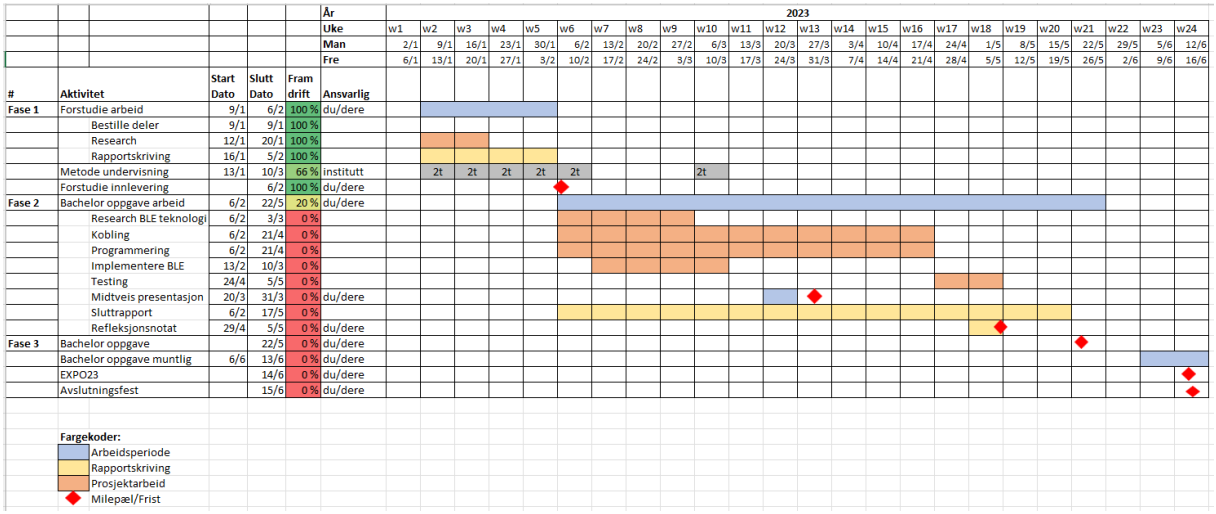
Arbeidet er fordelt likt på gruppemedlemmene, og vi har gjennomført de fleste oppgavene sammen. Alt arbeidet på prosjektet har blitt registrert i logg, i tillegg har vi begge ført timeliste hvor arbeidstimene som er gått til prosjektet ble registrert. Møter med veileder ble også loggført og ført inn som del av timeliste.

B.2 Git – Samarbeid på kode

Git er et verktøy for versjonskontroll som gjør det enkelt å holde oversikt over endringer i koden, og muliggjør at gruppemedlemmer kan arbeide med ulike deler av koden på samme tid. Når man jobber sammen på prosjekter som dette er det et nærmest essensielt verktøy. GitHub er den web-baserte plattformen hvor man laster opp koden i en såkalt «Git Repository», som også har vært brukt i forbindelse med dette prosjektarbeidet[4].

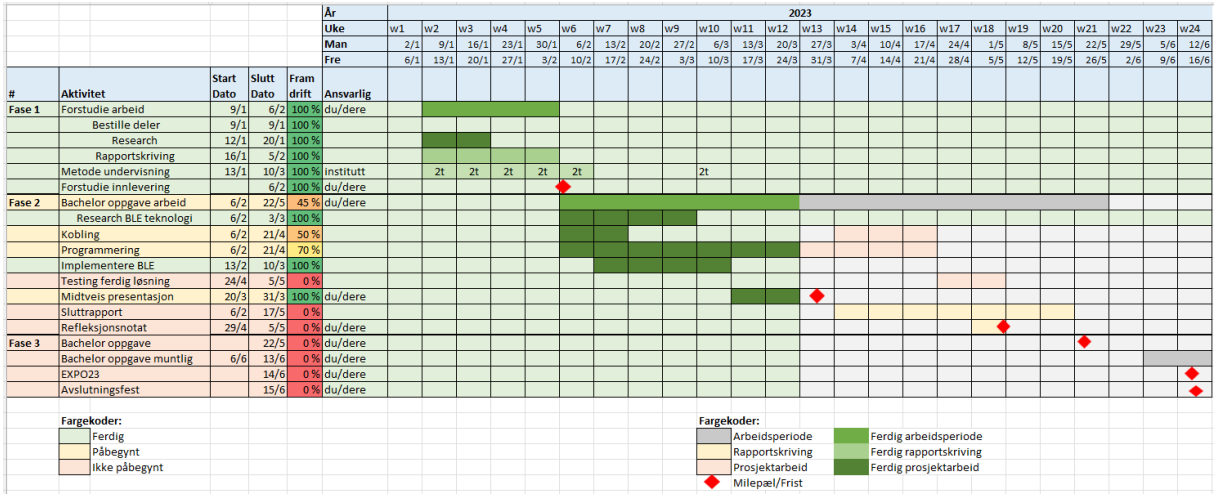
[4] https://github.com/stinehopland/BluetoothTemperatureRegulation

B.3 Fremdriftsplan



#	Aktivitet	Start Dato	Slutt Dato	Fram drift	Ansvarlig
Fase 1	Forstudie arbeid	9/1	6/2	100 %	du/dere
	Bestille deler	9/1	9/1	100 %	
	Research	12/1	20/1	100 %	
	Rapportskriving	16/1	5/2	100 %	
	Metode undervisning	13/1	10/3	66 %	institutt
	Forstudie innlevering		6/2	100 %	du/dere
Fase 2	Bachelor oppgave arbeid	6/2	22/5	20 %	du/dere
	Research BLE teknologi	6/2	3/3	0 %	
	Kobling	6/2	21/4	0 %	
	Programmering	6/2	21/4	0 %	
	Implementere BLE	13/2	10/3	0 %	
	Testing	24/4	5/5	0 %	
	Midtveis presentasjon	20/3	31/3	0 %	du/dere
	Sluttrapport	6/2	17/5	0 %	
	Refleksjonsnotat	29/4	5/5	0 %	du/dere
Fase 3	Bachelor oppgave		22/5	0 %	du/dere
	Bachelor oppgave muntlig	6/6	13/6	0 %	du/dere
	EXPO23		14/6	0 %	du/dere
	Avslutningsfest		15/6	0 %	du/dere

Figur 18: Fremdriftsplan Forstudie



#	Aktivitet	Start Dato	Slutt Dato	Fram drift	Ansvarlig
Fase 1	Forstudie arbeid	9/1	6/2	100 %	du/dere
	Bestille deler	9/1	9/1	100 %	
	Research	12/1	20/1	100 %	
	Rapportskriving	16/1	5/2	100 %	
	Metode undervisning	13/1	10/3	100 %	institutt
	Forstudie innlevering		6/2	100 %	du/dere
Fase 2	Bachelor oppgave arbeid	6/2	22/5	45 %	du/dere
	Research BLE teknologi	6/2	3/3	100 %	
	Kobling	6/2	21/4	50 %	
	Programmering	6/2	21/4	70 %	
	Implementere BLE	13/2	10/3	100 %	
	Testing ferdig løsning	24/4	5/5	0 %	
	Midtveis presentasjon	20/3	31/3	100 %	du/dere
	Sluttrapport	6/2	17/5	0 %	
	Refleksjonsnotat	29/4	5/5	0 %	du/dere
Fase 3	Bachelor oppgave		22/5	0 %	du/dere
	Bachelor oppgave muntlig	6/6	13/6	0 %	du/dere
	EXPO23		14/6	0 %	du/dere
	Avslutningsfest		15/6	0 %	du/dere

Figur 19: Fremdriftsplan Midtveis

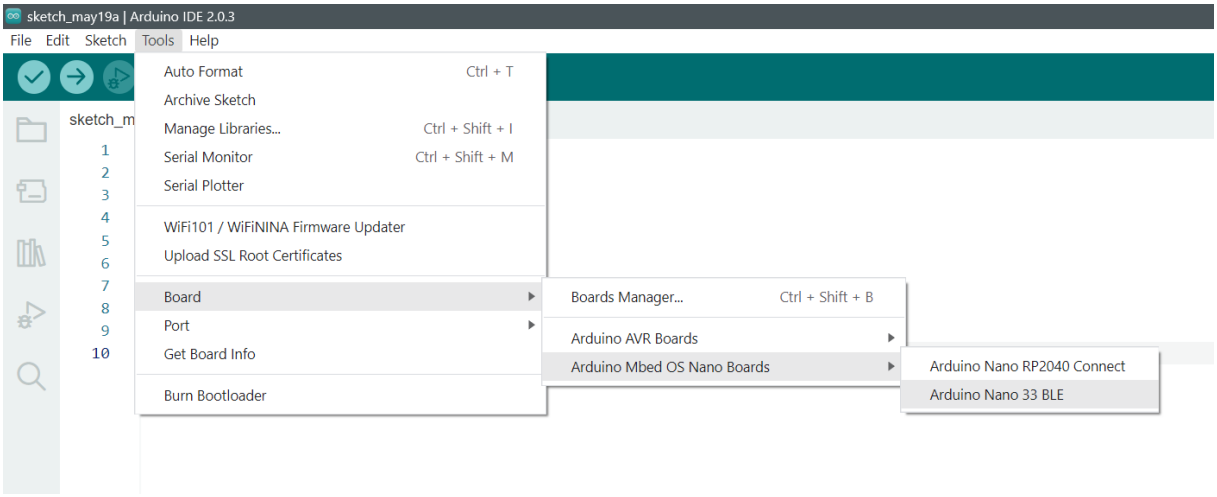
Appendiks C Brukerdokumentasjon

C.1 Brukerveiledning for programvare og utviklingsmiljø

Dersom brukeren ønsker å gjøre endringer i UWP-applikasjonen eller Arduino programmet, må de aktuelle utviklingsmiljøene først være lastet ned på datamaskinen (Visual Studio og Arduino IDE). Deretter kan all kode lastes ned fra vårt GitHub-repository[5]: BluetoothTemperatureRegulation.

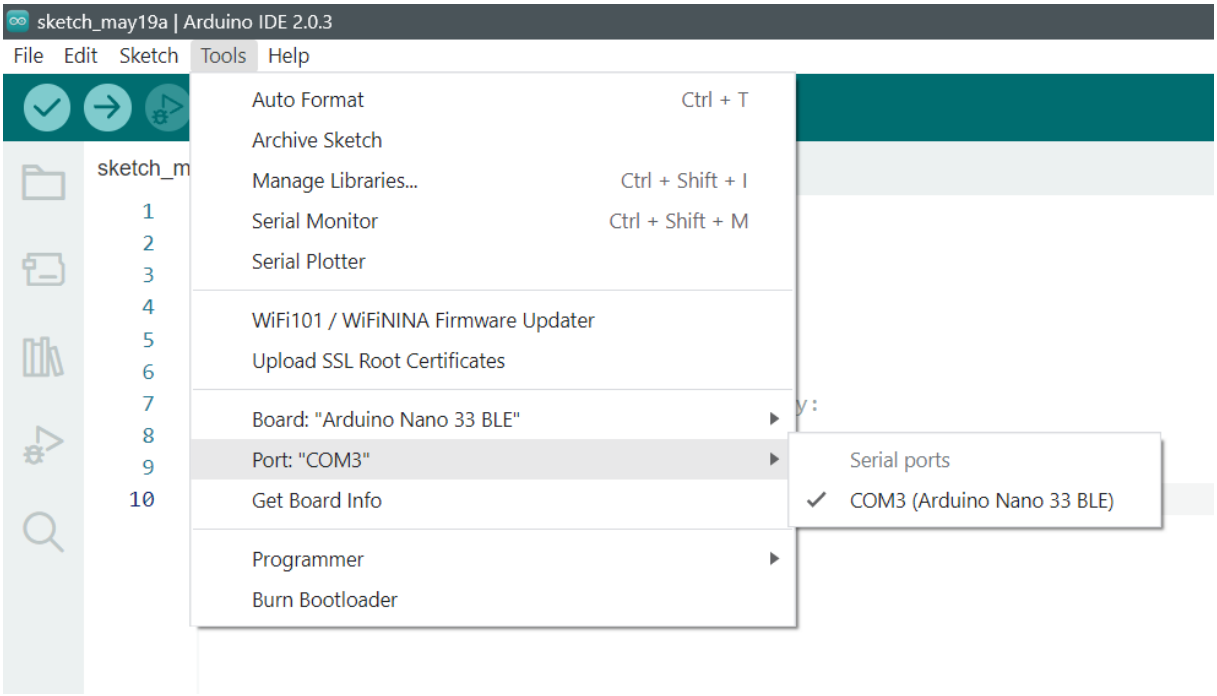
C.1.1 Arduino program

Det er mulig å gjøre endringer i programkoden for Arduino i flere ulike utviklingsmiljø, men vi anbefaler å benytte Arduino IDE. For å programmere Arduino må Arduino IDE åpnes, og Arduino enheten må være tilkoblet datamaskinen. I Arduino IDE kan Arduino-kortet velges under «Tools» > «Board», og riktig port velges under «Tools» > «Port». Dette er vist i henholdsvis figur 20 og 21.



Figur 20: Viser hvordan Arduino Nano 33 BLE kortet kan velges i Arduino IDE

[5] https://github.com/stinehopland/BluetoothTemperatureRegulation



Figur 21: Viser hvordan den aktuelle porten kan velges i Arduino IDE

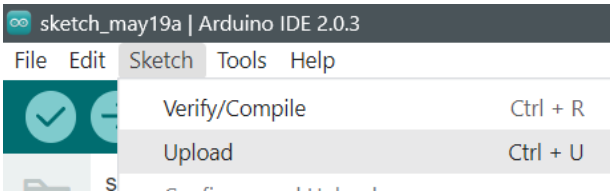
Den fullstendige programkoden fra denne oppgaven finnes i vår GitHub-repository[6], og kan nå åpnes og redigeres i Arduino IDE. Det er nødvendig å gjøre endringer i programmet eksempelvis dersom temperatursensoren skal byttes ut, eller at man ønsker å justere intervallet for avlesninger av data. Som vist i figur 22 er biblioteket SHT31 benyttet i koden, og den markerte linjen samt andre referanser i koden til «sht31» må byttes ut dersom en annen sensor skal benyttes. Figuren markerer også variabelen som er ansvarlig for det tidsintervallet som bestemmer hvor ofte Arduino leser temperaturdata fra sensoren.

6 https://github.com/stinehopland/BluetoothTemperatureRegulation/blob/master/Arduino/Temperaturregulering_ferdig.ino

```
//Library for Bluetooth Low Energy
#include <ArduinoBLE.h>

//Library for SHT31-temperatursensor
#include <SHT31.h>

//BLE Service UUID: Custom UUID
BLEService temperatureService("707c54e2-aff4-40ea-b80f-4ec2339425d3");

// BLE Temperature Characteristic UUID: Temperature Measurement
BLEFloatCharacteristic temperatureCharacteristic("2A1C", BLERead);

// BLE Characteristic UUID for Heat Element: Custom UUID
BLEBooleanCharacteristic heatElementCharacteristic("be3764b5-bcff-4e5c-a55b-ca698ef3cab7", BLEWriteWithoutResponse);


float temp = 0;
const unsigned HEATING_ELEMENT_PIN = D9; // pin used to control the heating element

unsigned long previousMillis = 0;
const long interval = 500; //intervall for avlesing

SHT31 sht31 = SHT31();
```

Figur 22: Markerer biblioteket som hører til SHT31 temperatursensor og intervallet som bestemmer hvor ofte Arduino leser av temperaturdata fra sensoren.

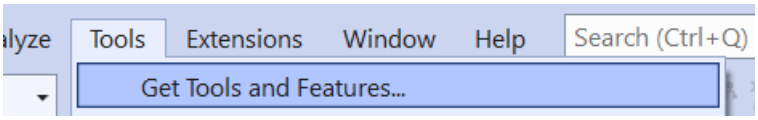
Etter de aktuelle endringene og tilpasningen har blitt gjort, må programmet kompileres og lastes opp til Arduino-kortet. I figur 23 ser vi hvordan dette kan gjøres enten ved å gå gjennom menylinjen, eller bruke hurtigtastene «Ctrl+R» og «Ctrl+U». Etter opplastningen er fullført, vil programmet med de nye endringene være lagret på Arduino-enheten.



Figur 23: Viser hvordan programkoden kompileres og lastes opp til Arduino-kortet i Arduino IDE

C.1.2 UWP utviklingsverktøy

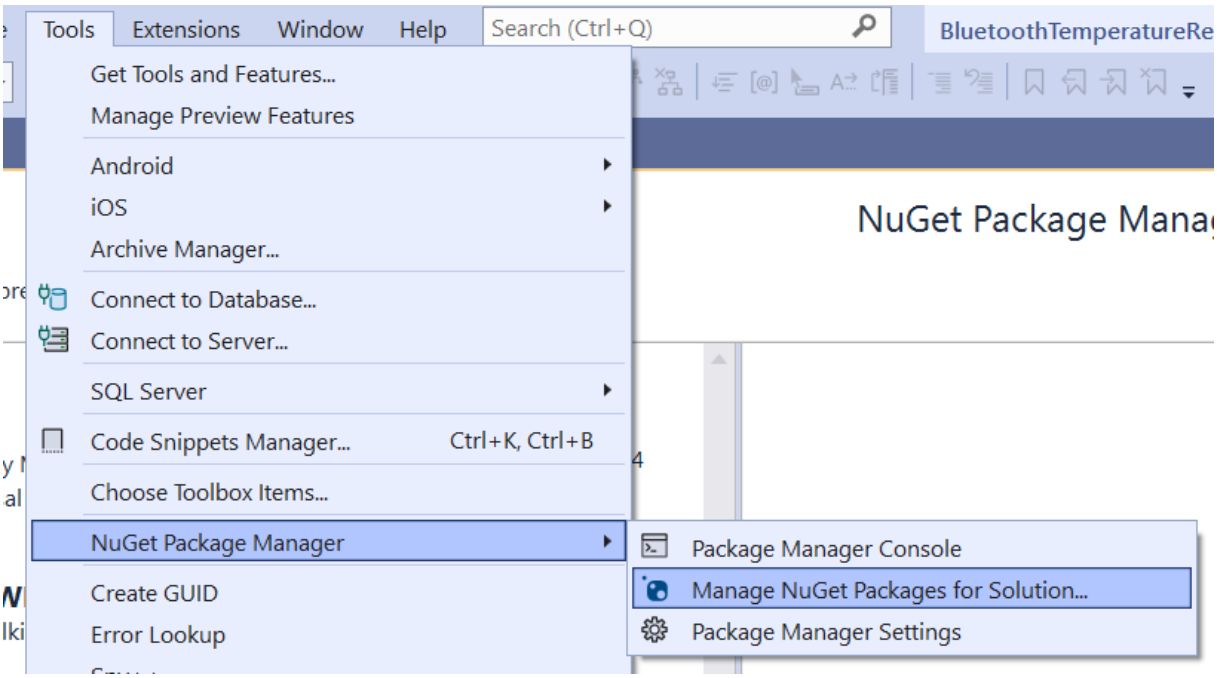
For å gjøre endringer i UWP-applikasjonen må man ha Visual Studio og «Universal Windows Platform development» installert på datamaskinen. Dersom vi tar utgangspunkt i at Visual Studio allerede er installert, kan man velge «Tools» > «Get Tools and Features» fra menylinjen (figur 24) for å sjekke at «Universal Windows Platform development» er huket av (figur 25). Hvis ikke, må denne installeres.



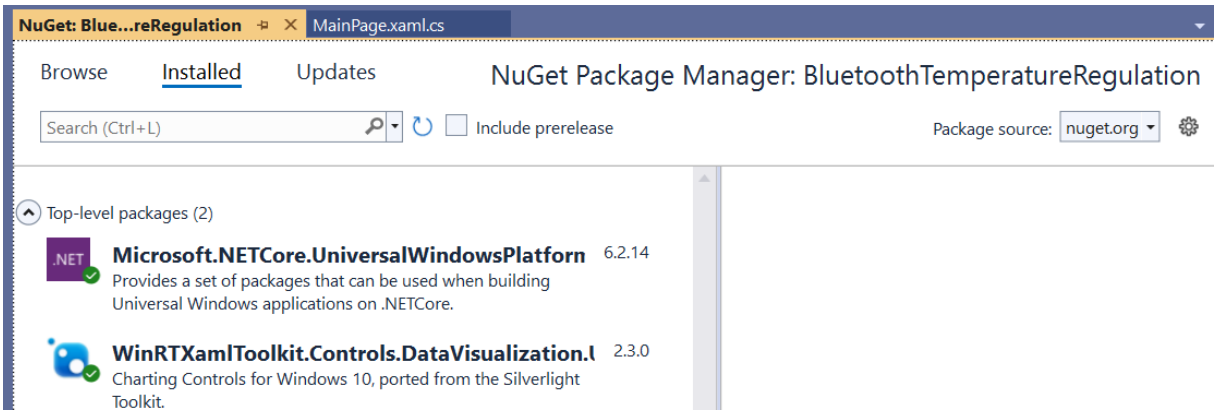
Figur 24: Viser hvor i menylinjen man må trykke for å finne ut hva som er installert.

Figur 25: Viser at Universal Windows Platform development er huket av og installert.

I programkoden har vi også vært avhengig av å laste ned NuGet-pakker som inkluder biblioteker og verktøy vi trengte for å få Bluetooth-funksjoner og Graf-funksjoner inn i applikasjonen. For å laste ned disse går man igjen i menylinjen og velger «Tools» > «NuGet Package Manager» > «Manage NuGet Packages for Solution…» som vist i figur 26. De pakkene vi installerte heter "*Microsoft.NETCore.UniversalWindowsPlatform*" og "*WinRTXamlToolkit.Controls.DataVisualization.UWP*", og er vist i figur 27.



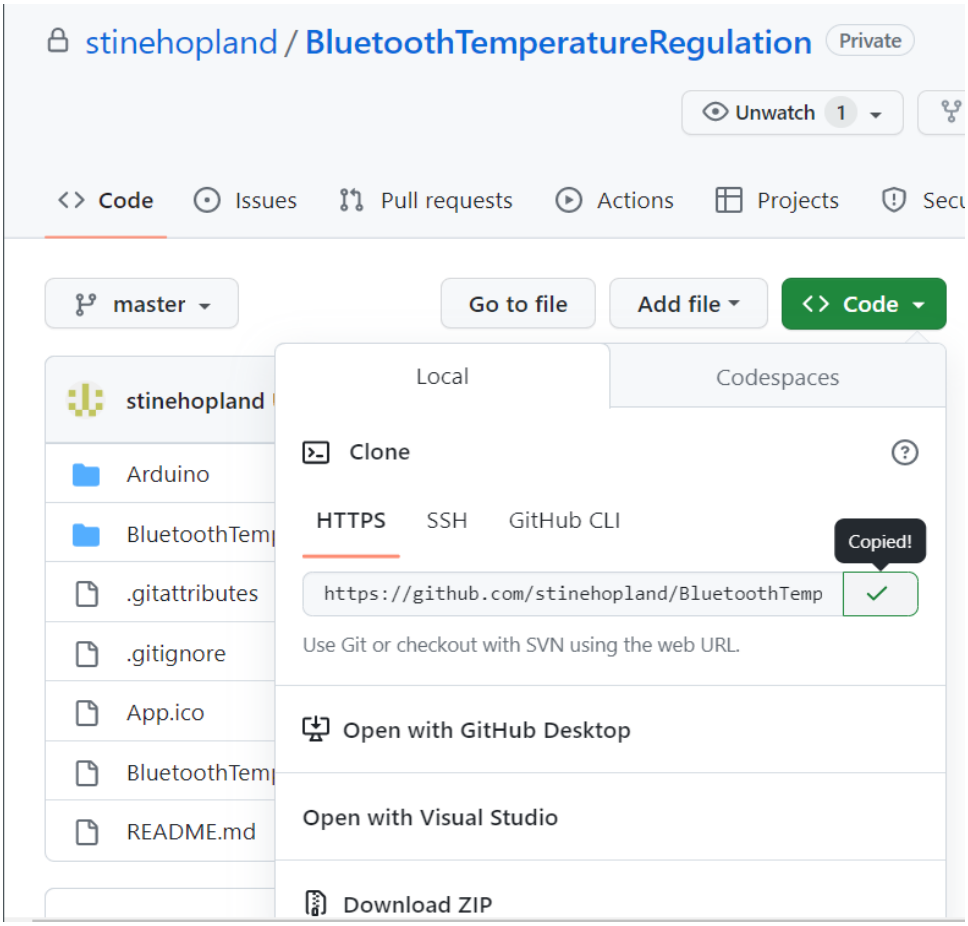
Figur 26: Viser hvor i menylinjen man kan installere NuGet-pakker



Figur 27: Viser de nedlastede NuGet-pakkene som er nødvendig å bruke i vår UWP-applikasjon.

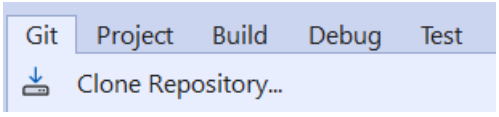
C.1.3 Utvikling av UWP-applikasjonen

For å videreutvikle eller gjøre endringer i UWP-applikasjonen, må man først ha fulgt stegene i C.1.2. Deretter laster man ned GitHub-repositoriet: BluetoothTemperatureRegulation, ved å laste ned ZIP-filen eller ved å klone hele repositoriet og åpne det i Visual Studio. Begge disse alternativene gjøres ved å følge linken til repositoriet, og deretter trykke på «Code» for å så kopiere linken for å klone eller trykke på «Download ZIP» dersom man ønsker det. Figur 28 viser hvordan dette ser ut.

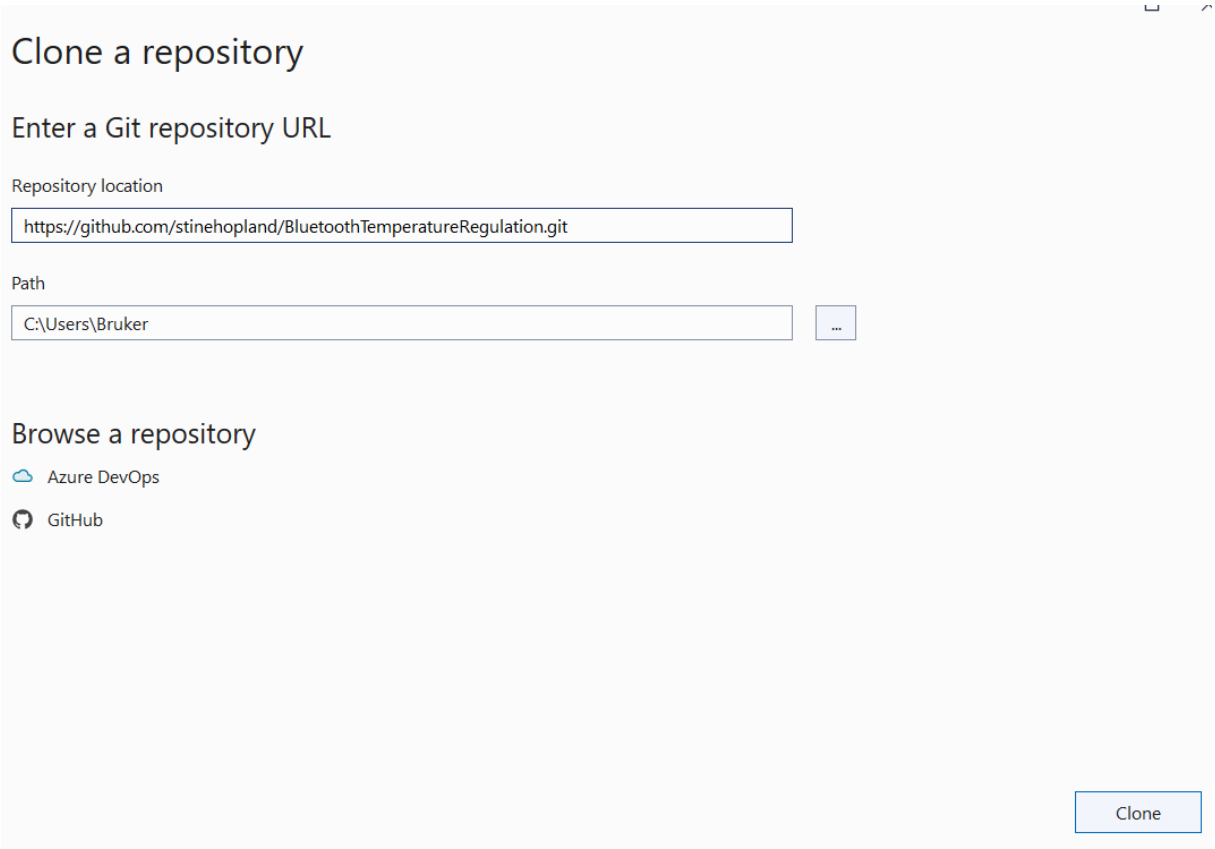


Figur 28: Viser hvor man kan klone eller laste ned ZIP-filen til repositoriet

Dersom man velger å klone GitHub-repositoriet for å så åpne det direkte i Visual Studio, kan man gjøre dette ved å velge «Git» > «Clone Repository» i menylinjen i Visual Studio, som vist i figur 29. Da får man opp det vinduet som er vist i figur 30, hvor klone-lenken som ble kopiert tidligere kan limes inn. Når disse stegene er gjennomført vil alle filene fra GitHub-repositoriet være tilgjengelig i Visual Studio, og man står fritt til å endre, videreutvikle og kjøre programmet fra Visual Studio.

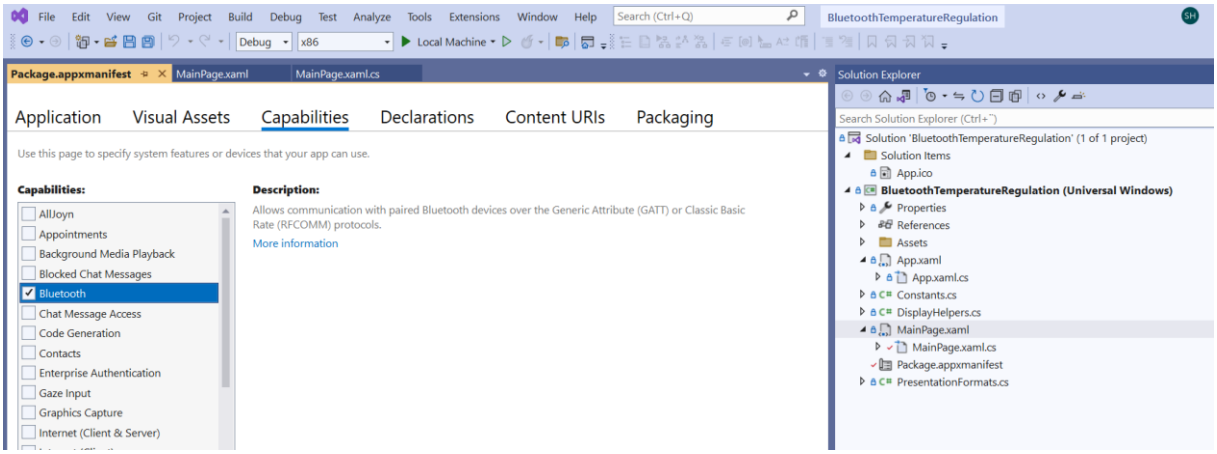


Figur 29: Menylinjen i Visual Studio.



Figur 30: Dette vinduet vises når "Clone Repository" velges fra menylinjen i Visual Studio. Her limes GitHub-repository URL'en inn og ønsket filplassering velges.

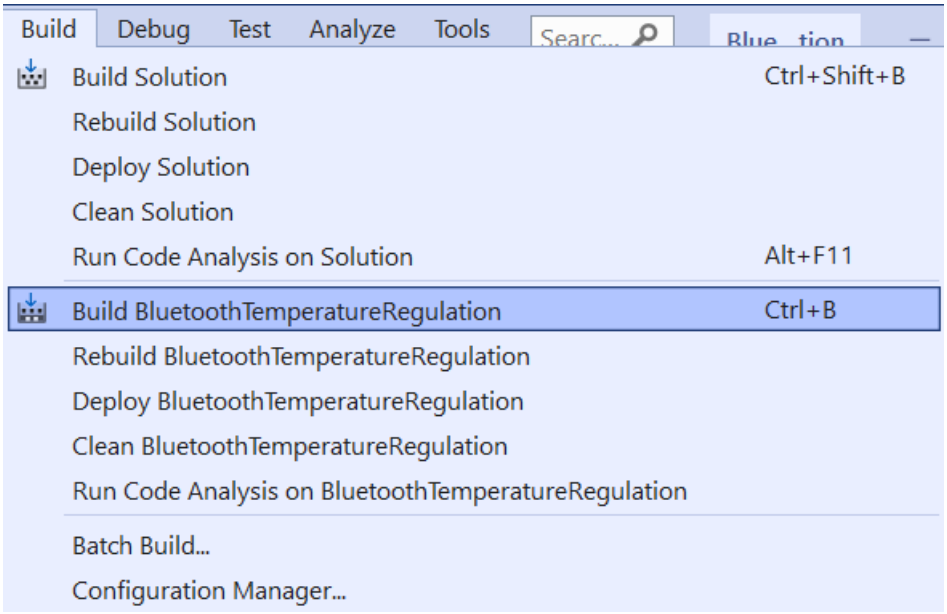
I tillegg til det overnevnte, er det viktig at 'Bluetooth' er huket av under «Package.appxmanifest» > «Capabilities», for at applikasjonen skal kunne benytte Bluetooth-kommunikasjon. Figur 31 viser hvordan det ser ut når 'Bluetooth' er huket av riktig.



Figur 31: Viser hvor 'Bluetooth' må hukes av.

C.2 Førstegangskjøring av UWP-applikasjon

Første gang man skal kjøre programmet på egen datamaskin må løsningen kompileres og det må genereres en kjørbar programvarefil fra koden. For å gjøre dette må man inn i menylinjen i Visual Studio og velge «Build» > «Build BluetoothTemperatureRegulation» og deretter «Build» > «Deploy BluetoothTemperatureRegulation», som vist i figur 32.



Figur 32: Viser hvor brukeren må trykke for å kompilere koden

Etter at løsningen har blitt kompilert og ingen feil er oppdaget i koden, kan man kjøre programmet. Dette gjøres ved å trykke på «Local Machine», som vises i figur 33.

Figur 33: Trykk på 'Local Machine' for å kjøre programmet

Appendiks D Software

D.1 Arduino Kode

Arduino programmet er skrevet spesifikt for bruk med en Arduino Nano 33 BLE og en Grove SHT31 temperatursensor, ettersom det bruker ArduinoBLE biblioteket for BLE kommunikasjon og SHT31-biblioteket til temperatursensoren. Programmet bruker BLE for å kommunisere med UWP-applikasjonen, og definerer UUID-ene for servicen og karakteristikkene som brukes for temperaturmåling og styring av varmeelementet. I oppsettfunksjonen starter programmet seriell kommunikasjon slik at man kan overvåke programmet fra 'Serial Monitor' i Arduino IDE, i tillegg til å sette opp pin-moduser, konfigurere BLE innstillingene og annonsere BLE servicen. I hoved løkken venter programmet på at UWP-applikasjonen skal koble til, og etter det starter programmet å lese temperaturverdien til sensoren og tilordne verdien til variabelen 'temp', samt oppdatere temperatur-karakteristikken med gjeldende verdi. Programmet starter også avlesing av varmeelement-karakteristikken, og setter den tilhørende pinnen til HIGH/LOW avhengig av om varmeelementet skal skrus av eller på.

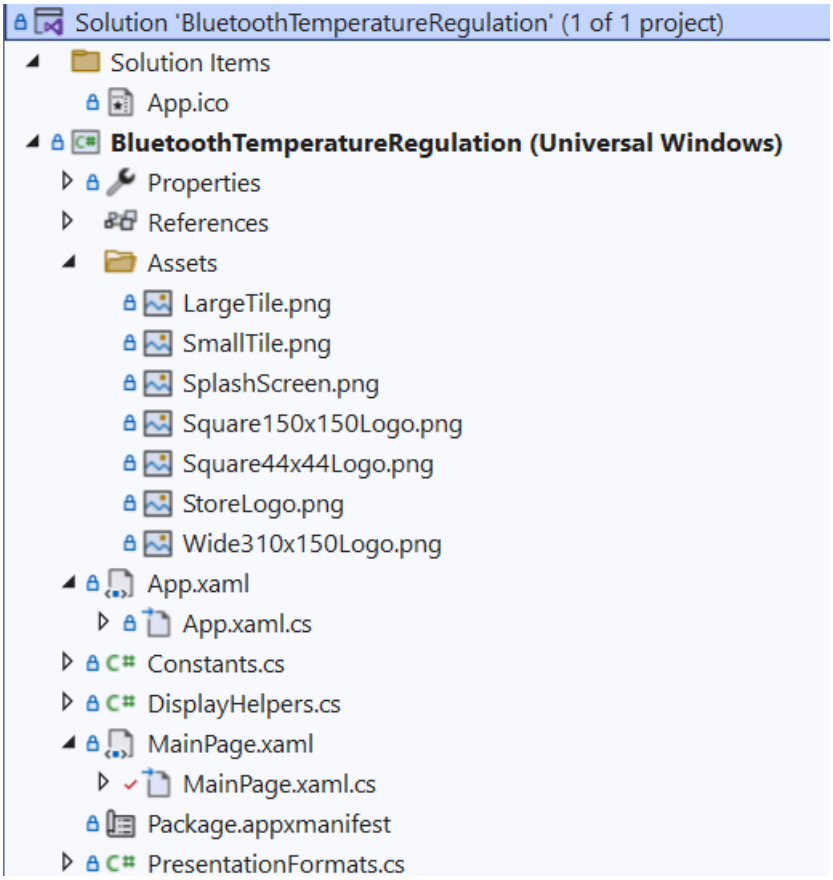
Nedenfor kan dere se utklipp av hele koden til Arduino programmet.

```
//Library for Bluetooth Low Energy
#include <ArduinoBLE.h>

//Library for SHT31-temperatursensor
#include <SHT31.h>

//BLE Service UUID: Custom UUID
BLEService temperatureService("707c54e2-aff4-40ea-b80f-4ec2339425d3");

// BLE Temperature Characteristic UUID: Temperature Measurement
BLEFloatCharacteristic temperatureCharacteristic("2A1C", BLERead);

// BLE Characteristic UUID for Heat Element: Custom UUID
BLEBooleanCharacteristic heatElementCharacteristic("be3764b5-bcff-4e5c-a55b-ca698ef3cab7", BLEWriteWithoutResponse);


float temp = 0;
const unsigned HEATING_ELEMENT_PIN = D9; // pin used to control the heating element

unsigned long previousMillis = 0;
const long interval = 500; //intervall for avlesing

SHT31 sht31 = SHT31();

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  // set Heating Element and built in LED pin to output mode
  pinMode(HEATING_ELEMENT_PIN, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.println("Begin...");
  sht31.begin();

  if (!sht31.begin()) {
    Serial.print("Temperature sensor not connected");
  }
  else {
    Serial.println("Temperature sensor connected");
  }
```

```
 // error message if Bluetooth does not begin
 if (!BLE.begin()) {
   Serial.println("Starting BLE failed!");
   while (1);
 }

 // set advertised local name and service UUID:
 BLE.setLocalName("Nano 33 BLE");
 BLE.setAdvertisedService(temperatureService);

 // add the characteristics of the service
 temperatureService.addCharacteristic(temperatureCharacteristic);
 temperatureService.addCharacteristic(heatElementCharacteristic);

 // add service
 BLE.addService(temperatureService);

 heatElementCharacteristic.writeValue(false);

 // start advertising
 BLE.advertise();
 Serial.print("Pheripheral device Bluetooth Address: ");
 Serial.println(BLE.address());
 Serial.println("Waiting for connections..");



oid loop() {
 // listen for BLE central to connect:
 BLEDevice central = BLE.central();

 // if a central is connected to peripheral:
 if (central) {
   Serial.println("Connected to central.");
   Serial.print("The clients MAC address is: ");
   // print the central's MAC address:
   Serial.println(central.address());
   Serial.print("Pheripheral device Bluetooth Address: ");
   Serial.println(BLE.address());
   digitalWrite(LED_BUILTIN, HIGH); // turn on LED to show it's connected



   // while central is connected to peripheral:
   while (central.connected()) {
     boolean heatElementValue = heatElementCharacteristic.value();

     if (heatElementValue != 0) { //any value other than 0
       Serial.println("Heat Element ON");
       digitalWrite(HEATING_ELEMENT_PIN, HIGH);  //turn on heating element
     }
     else {
       Serial.println("Heat Element OFF");
       digitalWrite(HEATING_ELEMENT_PIN, LOW); // turn off heating element
     }

     //assign the temperaturvalue from the sensor to the temp variable
     temp = sht31.getTemperature();

     //Print the value of the heating element characteristic to the serial monitor
     Serial.println(heatElementValue);

     //check if it is time to read and print temperature value
     unsigned long currentMillis = millis();
     if (currentMillis - previousMillis >= interval) {
       previousMillis = currentMillis;

       temperatureCharacteristic.writeValue(temp); //assign temperature value to characteristic
     }
   }

   // when the central disconnects:
   heatElementCharacteristic.writeValue(false);
   digitalWrite(LED_BUILTIN, LOW);
   digitalWrite(HEATING_ELEMENT_PIN, LOW); // turning off heating element
   Serial.print(F("Disconnected from central: "));
   Serial.println(central.address());
 }
```

D.2 UWP-applikasjonen

Koden til UWP-applikasjonen er skrevet i programmeringsspråkene C# og XAML, og kompilert i Visual Studio 2022. Programkoden følger den generelle oppbyggingen av en UWP-applikasjon, med en «solution» som fungerer som det overordnede prosjektet som inneholder alle filene, og deretter individuelle prosjektfiler, ressursfiler, konfigurasjonsfiler og kildekodefiler. I figur 34 kan man se en oversikt over de ulike filene og mappene i applikasjons-strukturen.



Figur 34: Oversikt over de ulike filene og mappene i appstrukturen.

Mappen 'Solution Items' inneholder app.ico, som er det ikonet som representerer applikasjonen, og 'Assets' mappen inneholder alle bilde-filene som brukes i applikasjonen.

Klassene 'Constants.cs', 'DisplayHelpers.cs' og 'PresentationFormats.cs' er hjelpeklasser som inneholder konstanter, variabler og funksjoner som kan brukes i hele applikasjonen.

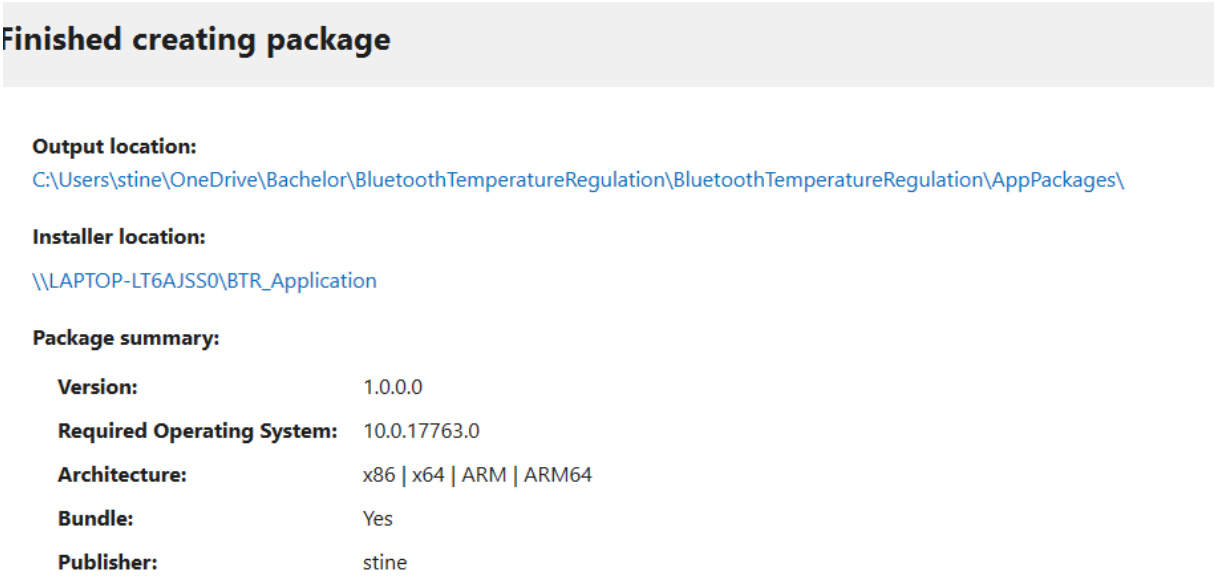
'App.xaml' og 'App.xaml.cs' definerer applikasjonens ressurser og oppstartoppførsel, og det er «.cs» filen som inneholder C#-koden som håndterer oppstartslogikken og applikasjonen sin livssyklusbehandling. Det er i denne filen vi initierer «Extended Execution», altså funksjonen som tillater applikasjonen å kjøre selv om den er minimert av brukeren [15].

Hoveddelen av programkoden er skrevet i 'MainPage.xaml' og 'MainPage.xaml.cs', og det er disse filene som definerer og styrer brukergrensesnittet. Førstnevnte er en XAML-fil som inneholder all koden bak hvordan brukergrensesnittet ser ut, og sistnevnte inneholder koden som bestemmer hvordan brukergrensesnittet skal oppføre seg.

'Package.appxmanifest' filen inneholder metadata og konfigurasjoner for applikasjonen, som blant annet tilgangen til Bluetooth.

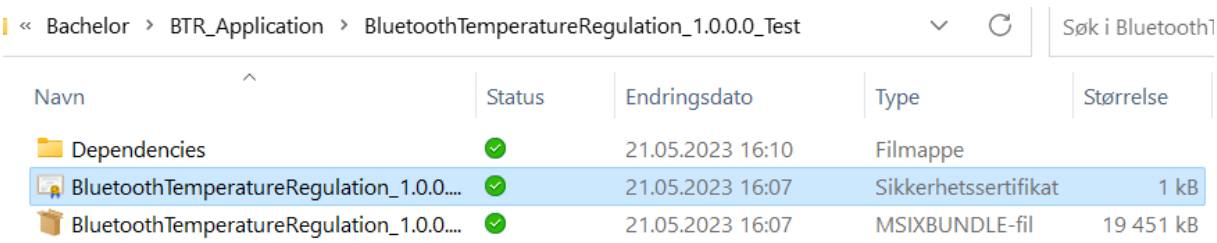
D.3 Installasjon av BluetoothTemperatureRegulation applikasjonen

Som nevnt tidligere, kan all kildekode sees og lastes ned fra GitHub, men vi har også laget en pakkefil som inneholder alle komponentene i appen. Denne pakkefilen muliggjør at applikasjonen enkelt kan distribueres for internt bruk, uten at det er nødvendig å legge ut appen i Microsoft Store. Pakkefilen er lastet opp i en delelig OneDrive mappe, og kan enkelt lastes ned på hvilken som helst datamaskin som kjører Windows 10 eller nyere. I figur 35 kan dere se en oversikt over denne pakkefilen, inkludert nødvendig operativt system og lokasjon for nedlastning.

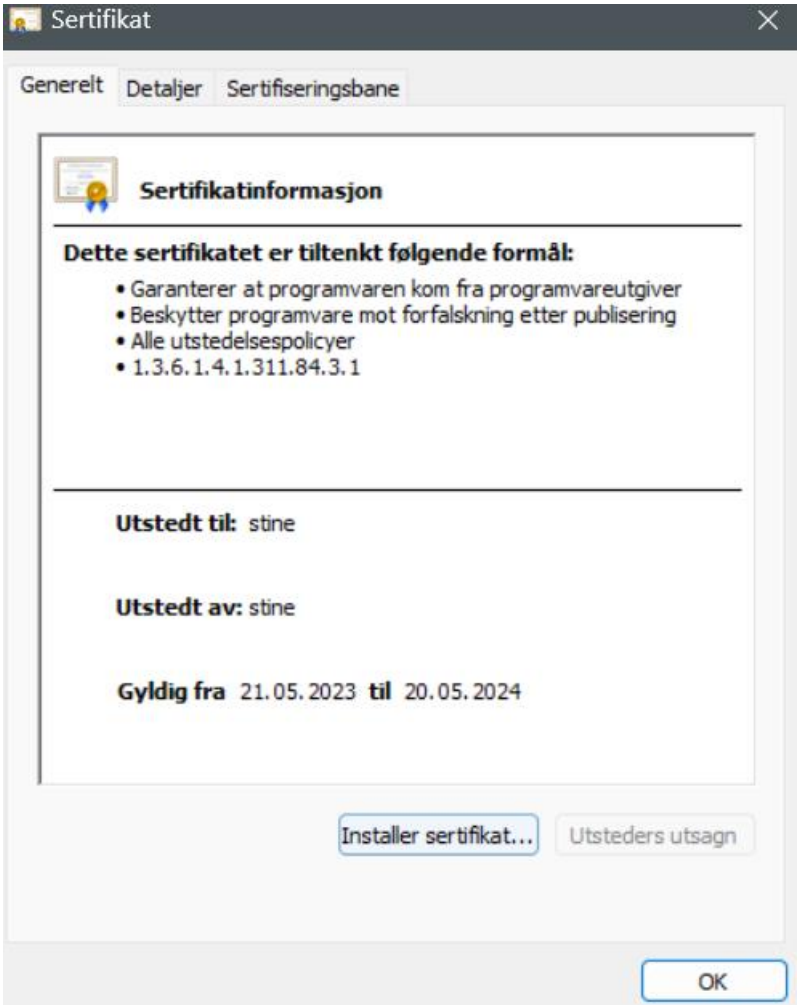


Figur 35: Oversikt over pakkefilen.

Figur 36 viser inneholdet i mappen «BTR_Application», og her ligger både 'appinstaller' filen og mappen «BluetoothTemperatureRegulation_1.0.0.0_Test» som inneholder alle de nødvendige filene, samt et sikkerhetssertifikat. Før applikasjonen kan fullføre installasjonen, må brukeren installere dette sertifikatet. Dette gjøres ved å dobbeltrykke på 'sertifikat'-filen. Passordet til dette sertifikatet er 'Bluetooth'.



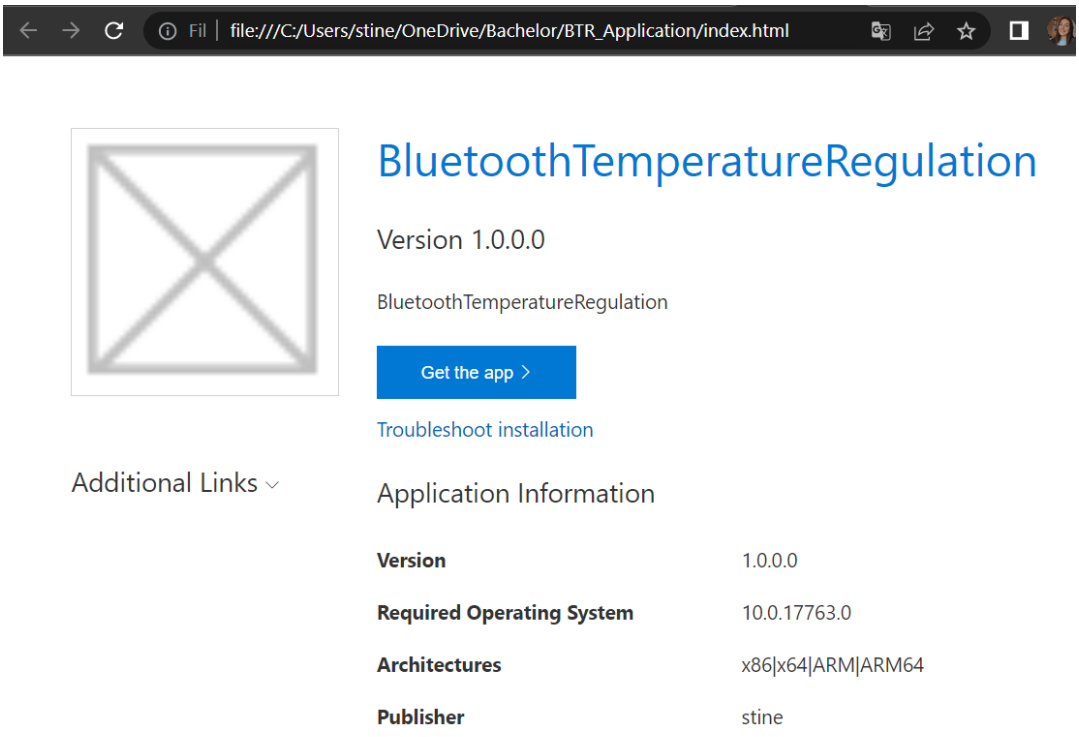
Figur 36: Sikkerhetssertifikatet må installeres.



Figur 37: Installasjonsvindu for sertifikat.

Etter at sertifikatet er installert vil det være mulig å installere applikasjonen. Dette kan gjøres ved å dobbeltrykke på '.appinstaller'-filen som vist i figur 37. Et annet alternativ er å følge index.html linken og installere applikasjonen via nettleseren, som vist i figur 38.

Figur 38: Installasjonsvindu for applikasjon.



Figur 39: Mulighet for å installere applikasjonen via nettleseren.

Appendiks E Hardware Komponenter

E.1 Arduino Nano 33 BLE

Vi har benyttet en Arduino Nano 33 BLE, ettersom denne bygger på en nRF52840 mikrokontroller som støtter Bluetooth Low Energy ekstern protokoll stack via BLE modulen NINA-B306 [23]. Denne mikrokontrolleren opererer med 3.3V input/output spenning til sine analoge og digitale pins, og får strøm fra en mikro-USB inngang som tåler 5-18V [16].

Hovedspesifikasjoner:

- nRF52840 mikrokontroller
- Micro USB connector m/ 5-18V nominell input spenning
- Nina B306 modul m/ Bluetooth Low Energy 5-multiprotokoll radio
- I2C kommunikasjon på pins: A4 (SDA) og A5 (SCL)
- 3,3 V I/O spenning

Pin-oversikt:

4.1 USB

Pin	Function	Type	Description
1	VUSB	Power	Power Supply Input. If board is powered via VUSB from header this is an Output **(1)**
2	D-	Differential	USB differential data -
3	D+	Differential	USB differential data +
4	ID	Analog	Selects Host/Device functionality
5	GND	Power	Power Ground

4.2 Headers

The board exposes two 15 pin connectors which can either be assembled with pin headers or soldered through castellated vias.

Pin	Function	Type	Description
1	D13	Digital	GPIO/Built-in LED
2	+3V3	Power Out	Internally generated power output to external devices
3	AREF	Analog	Analog Reference; can be used as GPIO
4	A0/DAC0	Analog	ADC in/DAC out; can be used as GPIO
5	A1	Analog	ADC in; can be used as GPIO
6	A2	Analog	ADC in; can be used as GPIO
7	A3	Analog	ADC in; can be used as GPIO
8	A4/SDA	Analog	ADC in; I2C SDA; Can be used as GPIO **(1)**
9	A5/SCL	Analog	ADC in; I2C SCL; Can be used as GPIO **(1)**
10	A6	Analog	ADC in; can be used as GPIO
11	A7	Analog	ADC in; can be used as GPIO
12	VUSB	Power In/Out	Normally NC; can be connected to VUSB pin of the USB connector by shorting a jumper
13	RST	Digital In	Active low reset input (duplicate of pin 18)
14	GND	Power	Power Ground

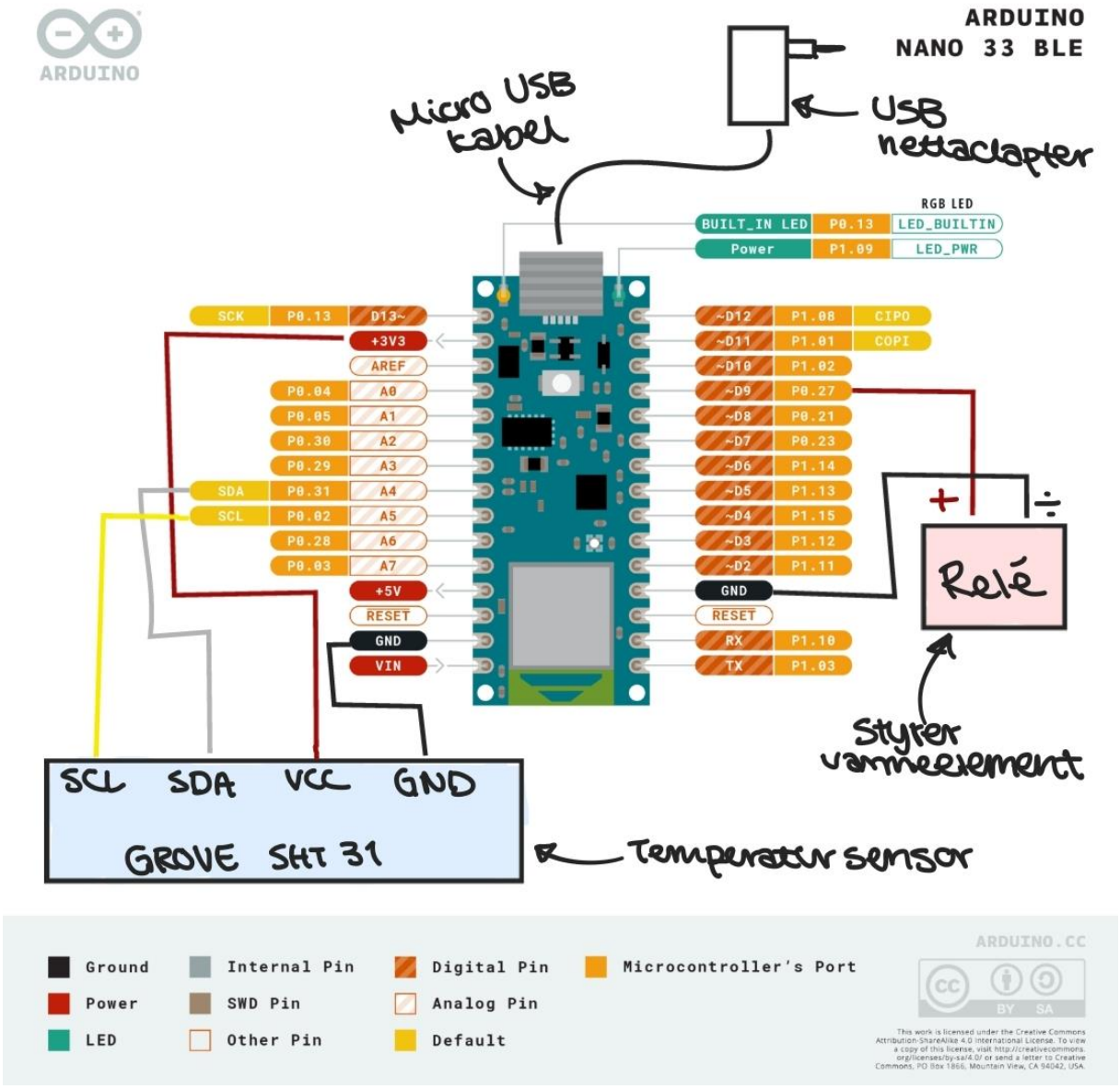
Figur 40: Oversikt over Arduino Nano 33 BLE sine power/analoge pins

Pin	Function	Type	Description
15	VIN	Power In	Vin Power input
16	TX	Digital	USART TX; can be used as GPIO
17	RX	Digital	USART RX; can be used as GPIO
18	RST	Digital	Active low reset input (duplicate of pin 13)
19	GND	Power	Power Ground
20	D2	Digital	GPIO
21	D3/PWM	Digital	GPIO; can be used as PWM
22	D4	Digital	GPIO
23	D5/PWM	Digital	GPIO; can be used as PWM
24	D6/PWM	Digital	GPIO, can be used as PWM
25	D7	Digital	GPIO
26	D8	Digital	GPIO
27	D9/PWM	Digital	GPIO; can be used as PWM
28	D10/PWM	Digital	GPIO; can be used as PWM
29	D11/MOSI	Digital	SPI MOSI; can be used as GPIO
30	D12/MISO	Digital	SPI MISO; can be used as GPIO

Figur 41: Oversikt over Arduino Nano 33 BLE sine digitale pins & VIN

Spesifikasjonene i figur 40 og 41 er hentet fra Arduino Docs og databladet til Arduino Nano 33 BLE [16] [24].

Arduino blir brukt til å behandle data fra temperatursensoren, styre varmeelementet samt å initiere BLE kommunikasjonen via programkoden som blir skrevet i Arduino IDE. Figur 42 viser hvordan de ulike komponentene er koblet til sine respektive pins på Arduino, ved hjelp av et pinout diagram laget av Arduino [17].



Figur 42: Koblingsskjema for Arduino, temperatursensor og relé. Bildet er et pinout diagram hentet fra: Arduino. Lisens: Creative Commons Attribution-ShareAlike 4.0 internasjonal lisens[7].

Temperatursensoren har fire koblingspunkter: SCL, SDA, VCC og GND. Som vi ser i figur 42 er SCL og SDA portene tydelig merket på Arduino. SCL og SDA koblingspunktene på sensoren kobles sammen med de to digitale pinsene på Arduino som er merket med henholdsvis SCL og SDA. Det er disse pinsene som muliggjør kommunikasjonen mellom sensoren og Arduino, ettersom de har støtte for kommunikasjonsprotokollen I2C som brukes til seriell dataoverføring. SDA brukes til selve dataoverføringen mellom enhetene, mens SCL brukes til å synkronisere overføringene [18]. VCC og GND koblingspunktene på sensoren kobles også til henholdsvis +3V3 (Voltage OUT) og GND på

[7] https://creativecommons.org/licenses/by-sa/4.0/

Arduino, og de sikrer strømtilførsel til sensoren. Vi velger å koble GND til på samme side som resten av pinnene som går til temperatursensoren, og VCC kobles til utgangen med 3,3 V fordi det er den spenningen sensoren krever. Reléet som skal styre varmeelementet kan kobles på en hvilken som helst digital pin, og oversikten over disse kan sees på figur 41. Vi har valgt å koble denne på D9 og bruker GND på samme side som denne porten for å gjøre det så ryddig som mulig. Arduino får strøm gjennom en mikro-USB kabel og USB nettadapter.

E.2 Temperatursensor

Temperatursensoren vi har benyttet i løsningen heter SHT31, og er en temperatur- og fuktighetssensor laget av produsenten Grove [6]. Sensoren har en nøyaktighet på +/-0.3 °C, og kan operere i temperaturer fra -40 til +125 °C [18]. Den er også kompatibel med 3.3 V og fungerer dermed til bruk sammen med Arduino Nano 33 BLE. Se figur 42 i appendiks E.1 for koblingsskjema.

Hovedspesifikasjoner:

- 3.3 V input spenning
- Kan operere i -40 til 125 °C
- Kan lagres i -40 til 150 °C
- +/-0.3 °C nøyaktighet

Pins:

- SDA – seriell data; I/O
- SCL – seriell klokke; I/O
- VCC – input spenning
- GND - jord

Konvertering av output signal:

Målt data blir overført gjennom de digitale pinsene via I2C-kommunikasjon i form av 16-bit verdier. For å konvertere disse verdiene til temperaturverdier, kan følgende formel benyttes:

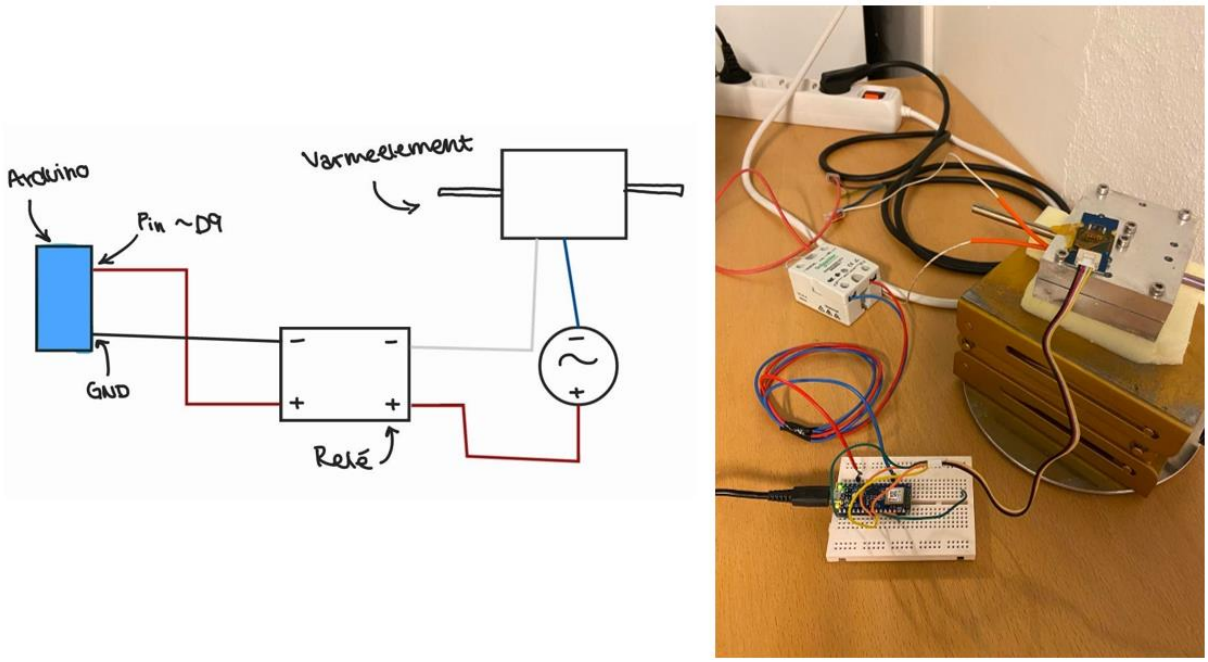
$$T\,[°C] = -45 + 175 * \frac{S_T}{2^{16} - 1}$$

S_T står for 16-bits rådataen fra sensoren. Formelen er hentet fra databladet til sensoren [18].

Denne sensoren er kun planlagt til bruk i utvikling og testing av løsningen og programvare. Temperatursensoren vil bli byttet ut av oppdragsgiver når programmet skal tas i bruk senere ettersom SHT31 ikke tåler de temperaturene som systemet og oppsettet på FARLAB krever.

E.3 Solid-State Relé

Arduino Nano 33 BLE sine digitale utganger har bare 3,3 V spenning. Varmeelementet vi skal bruke trenger 230 V AC. Vi bruker derfor et solid-state relé for å skru av og på varmeelementet. Reléet kan styres med 3 V og ha en utgangsspenning på 230 V [19]. Måten dette fungerer på er at det digitale signalet vi sender fra Arduino åpner og lukker en bryter som styrer utgangen på reléet og gir strøm til varmeelementet.



Figur 43: Til venstre: Koblingsskjema for kobling mellom Arduino Nano 33 BLE, relé og varmeelement. Til høyre: Bilde av fysisk kobling for testing av system med varmeelement og temperatursensor (Bilde tatt på Universitetet i Bergen utenfor FARLAB 25.04.2023)

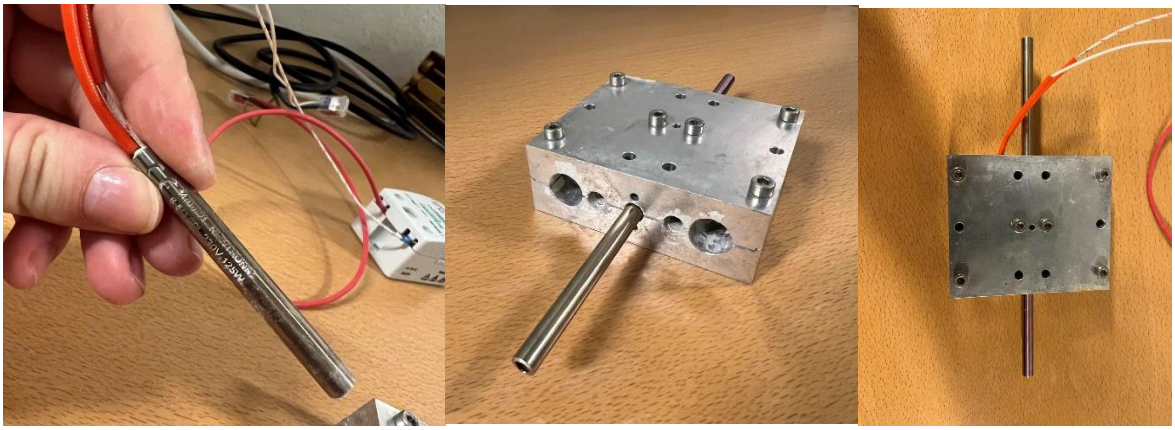
På figur 43 kan man se koblingsskjema for koblingen mellom Arduino, relé og varmeelement, og bilde av den ferdige fysiske oppkoblingen med temperatursensor i tillegg. Reléet skal styres av Arduino og er koblet mellom den digitale utgangen D9 på jord (GND) på selve Arduino brettet. Utgangen av reléet kobles til 230 V AC strømforsyning gitt via strømkabel koblet i stikkontakt. Varmeelementet kobles mellom strømforsyningen og minuspolen på utgangen til reléet.

E.4 Varmeelement

Varmeelementet vi brukte for testing av programvaren vår består av en patronvarmer og en metallblokk. En patronvarmer er et rørformet industrielt varmeelement som kan settes inn i borede hull. Patronvarmere gir lokalisert og presis oppvarming og er vanligvis brukt i oppvarmings-prosessindustrien, og brukt for å varme metallblokker fra innsiden. En patronvarmer består av en motstandsspole viklet rundt en keramisk kjerne som er omgitt av dielektrum, altså et isolerende eller meget dårlig ledende materiale brukt i sammenheng der det utsettes for et elektrisk felt, og innkapslet

i en metallkappe [20]. Når patronvarmeren får tilført strøm gjennom spolen vil det produseres varme som overføres til metallkappen og videre til metallblokken patronvarmeren sitter i [21].

Patronvarmeren vi brukte har en effekt på 125 W og bruker 230 V AC. Metallblokken er laget av aluminium. Aluminium er brukt fordi det raskt varmes opp når det blir tilført varme, og raskt kjøles ned igjen når man slutter å tilføre varme. Man kunne brukt kobber, gull eller sølv også. Dette ville gjort oppvarming og nedkjølingsprosessen raskere og temperaturreguleringen ville blitt mer nøyaktig, men det er mye dyrere og aluminium fungerer mer enn godt nok når vi bare skal teste at programvaren fungerer som den skal.



Figur 44: Bilde av patronvarmer, metallblokk og ferdig sammensatt varmeelement med patronvarmer inni metallblokk (Bilde tatt utenfor FARLAB ved Universitetet i Bergen 16.05.2023)

På figur 44 kan du se bilde av patronvarmeren til venstre, metallblokken med hull i midten og det ferdig sammensatte varmeelementet til høyre med patronvarmeren plassert inni metallblokken.