Flexibilitet på efterfrågan för elektriska värmesystem

Rapport för kandidatarbete inom Elkraft

Arvid Erlandsson
Lucas Fosse
Lisa Holländer
Victor Rasmusson
Simon Skogsberg

CHALMERS TEKNISKA HÖGSKOLA
E2 – Institutionen för Elektroteknik
Göteborg, Sverige, 8 maj 2024

Förord

Vi vill tacka de som hjälpt oss under projektets gång och som därigenom gjort denna uppsats möjlig. Tack till vår handledare David Steen som stöttat och väglett oss under projektets gång, samt gett värdefull feedback på rapporten. Tack till vår examinator Jimmy Ehnberg för ett konstruktivt mittmöte. Slutligen vill vi rikta ett speciellt tack till Magnus Ellsén för hans snabba kommunikation, goda råd och handlingsförmåga. Utan honom hade vi inte kommit in i forskningsanläggningen, ännu mindre kunnat skriva om den i detalj.

Göteborg, 23:22 8 maj 2024.

Arvid, Lucas, Lisa, Victor, Simon.

Innehåll

Abstract

Financial incentives were introduced by the swedish government as part of an effort to encourage real estate owners to install solar panels, with the goal to lessen the burden on the electric grid whilst accelerating the transition to green energy. The purpose of this project is to develop a control system for heating that will lower the energy cost of a building. The control system will, by using forecasts for electricity production and the outside temperature, along with a thermal model of the building, be able to plan for next day's heating and optimize towards the lowest cost. This will be implemented on the research facility that Chalmers University of Technology owns on the island of Björkö, for test and demonstration purposes. Furthermore, historical data is needed in order to develop the forecasts and to create a thermal model of the building. Another aim of this project is therefore to implement a data collection system for the facility.

The optimization model and its corresponding parts for the control system were created using Python. The Python library Pyomo was used for the optimization, made possible by adapting the problem statement to equations according to the linear programming standard form. In order to create the forecast for solar production the already existing tools from the online website Rebase were used, while the forecasts for the price of electricity and the outdoor temperature were retrieved directly from ENSTO-E and SMHI. The thermal model was derived from an already existing model, and adapted to fit the specific building's thermal characteristics. Furthermore a single-board computer was installed in order to communicate with a heating fan connected to the research facility on Björkö. The heating fan was connected to the facility in order to control the heat of the building with input from the optimization model, in such a way that the fan could be switched on and off remotely.

The data collection system took more time than anticipated, which led to delays and resulted in less time to put on the later stages of the project. This led to difficulties testing and adapting the forecasts, the optimization model and the control system. This, coupled with discovering errors in the thermal model in the last weeks of the project, led to varying results. Despite of this, the results obtained from the tests show that the output from the optimization model used by the control system does steer the heating system towards the lowest cost. The conclusion of the project is that the optimization model and the control system does work, but more changes and testing is needed in order for the system to work optimally.

Sammanfattning

För att uppmuntra fastighetsägare att installera solpaneler introducerade den svenska regeringen ekonomiska incitament med målet att minska bördan på elnätet såväl som att påskynda omställningen till förnybar energi. Syftet med projektet är att utveckla ett kontrollsystem för uppvärmning som ska minska energikostnaderna för en byggnad. Kontrollsystemet ska, genom att använda prognoser för elproduktion, utomhustemperatur och en termisk byggnadsmodell, kunna planera för nästa dags uppvärmning och optimera mot den lägsta kostnaden. Detta kommer implementeras på forskningsanläggningen som Chalmers tekniska högskola äger på Björkö, i test och demonstrationssyften. Vidare behövs historiska data för att utveckla prognoserna och skapa en termisk modell av byggnaden. Därför är ett ytterligare syfte med projektet att implementera ett datainsamlingssystem för anläggningen.

Optimeringsmodellen utvecklades i programmeringsspråket Python. Pythonbiblioteket Pyomo användes för optimeringen, vilket var möjligt efter att ha omformulerat problemställningen till en målfunktion enligt standardformen för linjär programmering. Optimeringsmodellen nyttjade prognoserna, som tränades på data från byggnaden. För att skapa prognosen för solproduktion användes det redan existerande prognosverktyg från Rebase, medan prognoserna för elpris och temperatur togs direkt från ENSTO-E respektive SMHI. Den termiska modellen skapades från en redan befintlig mall, som anpassades till byggnadens termiska egenskaper. Vidare installerades en enkortsdator för att kommunicera med en värmefläkt ansluten till forskningsanläggningen på Björkö. Värmefläkten anslöts till anläggningen i syfte att kunna kontrollera värmen i byggnaden enligt värmeschemat från optimeringsmodellen, så att värmefläkten kunde fjärrstyras.

Datainsamlingssystemet tog mer tid än förväntat, vilket ledde till förseningar och brist på tid i senare delar av projektet. Det blev därför svårt att noggrant testa och anpassa prognoserna, optimeringsmodellen och kontrollsystemet. Detta, tillsammans med brister i den termiska modellen som upptäcktes i den senare delen av projektet, ledde till varierade resultat. Trots detta så visar resultaten från testerna att värmeschemat som kontrollsystemet erhåller från optimeringsmodellen faktiskt styr systemet mot den lägsta kostnaden. Slutsatsen av projektet är att optimeringsmodellen och kontrollsystemet fungerar, men att vissa ändringar och fler tester behövs för att systemet ska fungera optimalt.

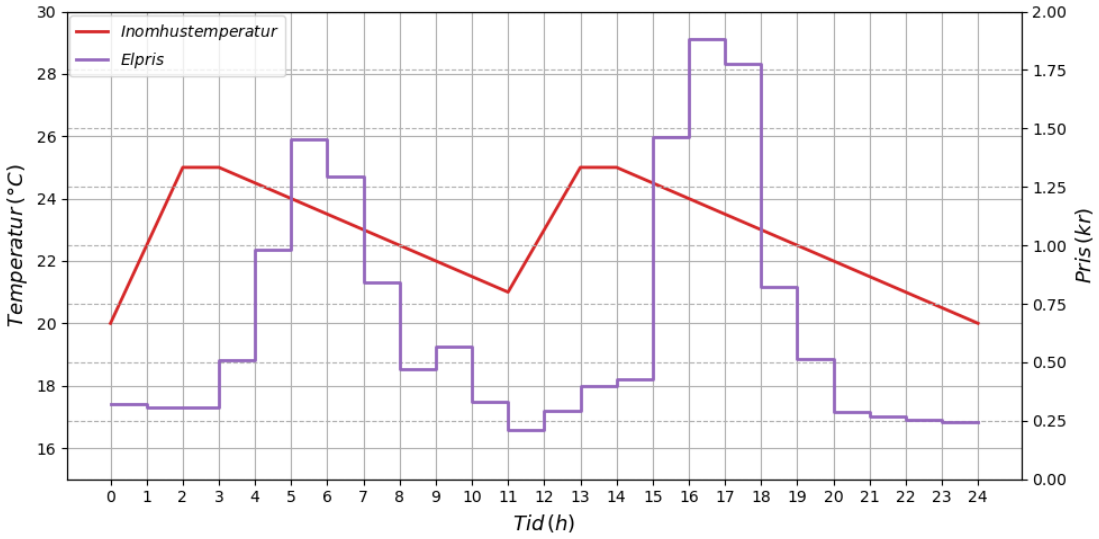
1 Inledning

Som en del av klimatarbetet, integreras det mer och mer förnyelsebara energikällor på elnätet [1]. Både solenergi och vindkraft är till naturen väderberoende. Väder är svårt att förutspå och förhålla sig till, och med en allt större andel väderberoende energikällor på elnätet, kommer energitillgången att bli mer variabel [1].

Att förhålla sig till en variabel energitillgång förutsätter att konsumenten också kan vara variabel i sin förbrukning, med andra ord flexibel [1]. En viktig förutsättning för att kunna vara flexibel är att energibehovet är någorlunda tidsoberoende. Ett illustrativt exempel är att ladda en elbil. I det användningsområdet är det viktiga att bilen är laddad när den ska köras, men mindre viktigt när den ska laddas. Medans människor kan anpassa sitt energibehov rätt väl i tid, så finns det automatiska system som inte kan anpassa sig lika lätt.

1.1 Bakgrund

Ett mer eller mindre automatiskt effektbehov, som besitter samma tidsoberoende egenskap som beskrivs tidigare, är uppvärmning av hus och anläggningar. Den energi som tillförs till en anläggning lämnar typiskt systemet i en mycket långsammare takt. Skulle uppvärmningen upphöra, dröjer det ofta ett tag innan kylan blir påtaglig. Reglersystem för många termostater är idag inte designade utifrån ett flexibelt perspektiv, trots potentialen till det. Fokus brukar vara att hålla en given temperatur, utan någon tidsuppfattning i regleringen. Det hade därför varit intressant att undersöka om det går att implementera tidsuppfattning i ett sådant kontrollsystem, och vilken data som är väsentlig för att fatta smarta beslut om värmebehovet. Detta för att se hur mycket nytta det finns att erhålla av ett flexibelt uppvärmningssystem. I figur 1 visas ett fiktivt exempel på hur ett mer flexibelt uppvärmningssystem hade kunnat se ut med avseende på elpriset.



Figur 1: Fiktivt exempel på hur uppvärmningen av en byggnad kan variera, fiktiv inomhustemperatur och elpris.

Elpriset är en bra uppskattning på hur energitillgången ser ut för olika timmar. Genom att värma en byggnad när elpriset är lågt hade det hypotetiskt gått att undvika pristoppar och följaktligen timmar med hög trafik på elnätet. Detta förutsätter vissa förhållanden, framförallt en termisk tröghet hos byggnaden som tillåter de kapacitiva förhållanden illustrerade i figur 1. Utöver detta ställer en sådan uppvärmning krav på en viss framförhållning vad gäller kritiska faktorer, exempelvis utomhustemperatur och andra meteorologiska variabler, information om energiproduktionen på anläggningen samt elpriset över en bestämd tidshorisont. Med detta i åtanke hade det varit fördelaktigt med ett kontrollsystem, för att vid en viss tidpunkt kunna ta i beaktande dessa omständigheter och planera för uppvärmningen. För ett sådant kontrollsystem krävs olika prognoser och modeller av byggnaden och till dessa information om byggnaden för att säkerställa en viss precision.

Chalmers bedriver mycket forskning kring olika aspekter av förnyelsebar elproduktion, bland annat lastflexibilitet. På Björkö, ute i Göteborgs skärgård, har högskolan en anläggning med de grundläggande komponenterna för forskning kring den gröna omställningen. Kontrollrummet värms idag upp av ett elektriskt element och uppvärmningen utgör majoriteten av anläggningens effektbehov. Det är alltså den perfekta miljön för att utforska frågeställningar kring lastflexibiliteten av ett sådant kontrollsystem för uppvärmning. En bild på forskningsanläggningens två byggnader visas i Figur 2.

Figur 2: Byggnaderna på anläggningen. Till vänster är fikabyggnaden. I byggnaden till höger finns energimätare, kraftelektronik och annan liknande utrustning.

För att utnyttja produktionsmöjligheterna på anläggningen och för att få en idé om den generella energikonsumtionen behövs någon form av datainsamling. På anläggningen finns tre energimätare: en för solcellerna, en för vindkraftverket och en för utrustningen. Mätarna saknar implementerade metoder för att extrahera data. Det enda sättet är att visuellt läsa av mätaren via en skärm. För att få översikt av anläggningens produktion och konsumtion samt fördelningen av denna hade det därför varit fördelaktigt att implementera hårdvara som kan tillåta kommunikation med dessa och samla in data.

1.2 Syfte

Syftet med projektet är att utveckla ett kontrollsystem som kan göra uppvärmningen mer flexibel. Detta kontrollsystem ska, med kännedom om elförbrukning och prognoser för elproduktion och utomhustemperaturen, kunna planera för morgondagens uppvärmning samt optimera för den lägsta kostnaden. Målet är att implementera detta system på anläggningen för tester och demonstrationer.

För att utveckla prognosmodeller och ta fram en termisk modell av byggnaden behövs historisk mätdata från anläggningen. Ytterligare ett mål för projektet är därför att implementera ett datainsamlingssystem för anläggningen.

1.3 Avgränsningar

För att projektet ska kunna hålla fokus, och inte omfatta allt för många tidskrävande moment, görs några avgränsningar. Fokus kommer inte läggas på design och utveckling av egen hårdvara för styrning, då detta hade varit tidskrävande och inte i linje med projektets syfte. Existerande hårdvarulösningar används därför för att uppnå projektets mål kring styrningen av uppvärmningen.

På forskningsanläggningen finns det två byggnader. En av dem huserar servrar och kontrollutrustning. Den andra byggnaden är fikarum för de som besöker anläggningen. Kontrollsystemet kommer bara implementeras i en av dessa: den förstnämnda, eftersom projektet behöver nära tillgång till utrustningen för datainsamling.

När optimeringsmodellen konstruerades och användes tillämpades en förenklad modell av elpriset i jämförelse med verkligheten. Elpriset (för ett timavtal som följer spotpriset) bestämts också av elskatten som enbart appliceras på importerad el. Detta gjordes med syfte att förenkla optimeringen och behandla priset likadant vid import och export.

Ytterligare en avgränsning görs med avseende på egenproduktionen. Vindkraftverket som finns på anläggningen körs endast sporadiskt vid experiment och demonstrationer. Det kan bli svårt att få meningsfull data från vindkraften då den sällan körs, så den prognosmodell som projektet tar fram för egenproduktion kommer inte att omfatta vindkraftverket utan enbart produktion från solcellsanläggningen.

Vidare krävs det en termisk modell för att avgöra värmedynamiken i byggnaden. Fokus i projektet ligger på optimering av kontrollsystemet och det finns god tillgång på färdiga termiska modeller, som går att anpassa genom mätningar för olika parametrar. Därför skapas inte en ny termisk modell för byggnaden, utan en befintlig modell anpassas till byggnaden.

Avslutningsvis behöver kontrollsystemet en prognos för anläggningens solproduktion. För denna kommer hemsidan Rebase, som tillåter modelldesign baserad på historisk data, att användas. Många delmoment i projektet hade kunnat vara arbeten i sig, exempelvis termisk modell och prognoser över solproduktionen, vilket är varför dessa avgränsningar görs.

2 Teori

Följande avsnitt beskriver teorin av vilket projektet är uppbyggt.

2.1 Optimering

Som tidigare nämnt är syftet med projektet att öka flexibiliteten för ett elektriskt värmesystem genom att minska energikostnaderna hos byggnaden projektet utgår ifrån. Ett annat sätt att se på det är att projektet ämnar att maximera flexibiliteten och minimera kostnaderna.

Den grenen av applicerad matematik som hanterar minimering och maximering av funktioner kallas optimering [2]. Optimering har utvecklats från att vara ett sätt att beskriva minimeringen av energifunktioner inom ämnen som mekanik och fysik, till ett studieområde under datavetenskap som behandlar applikationen av algoritmer för att lösa matematiska problem med hjälp av datorer [2]. Optimering är en teknologi som används för att skapa effektiva beslut och förutsägelser [2]. Processen för att nå beslutet börjar vid konstruktionen av en passande matematisk modell för ett konkret problem, och följs av en fas där modellen löses med hjälp av passande numeriska algoritmer [2]. En optimeringsmodell kräver att man specificerar ett kvantitativt objektivt kriterium för det önskade målet som ska maximeras eller minimeras, även kallat *målfunktion* [2]. Vidare behövs det specificeras bivillkor som beskriver de begränsningar av beslutet och dess verkan. Dessa bivillkor kan exempelvis bestå av en budget eller kostnad för resurser, eller rent designmässiga aspekter. En optimering kommer ge det bästa objektiva värdet, inom ramen av de villkor som ställts [2]. Dessa samband kan beskrivas enligt följande matematiska funktioner enligt (7.1.1) i [3].

$$\text{Minimera:} \quad f(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^n \tag{2.1}$$

$$\text{Givet att:} \quad c_i(\mathbf{x}) = 0, \; i \in E \tag{2.2}$$

$$c_i(\mathbf{x}) \geq 0, \; i \in I \tag{2.3}$$

Målfunktionen är $f(\mathbf{x})$, och bivillkoren $c_i(\mathbf{x})$, $i = 1, 2, 3, ..., p$. E är mängden av index för ekvationerna eller likhetsvillkoren i problemet, I är mängden av olikhetsvillkor, och båda dessa är ändliga [3].

2.1.1 Linjär programmering

Det mest enkla uttrycket för ett optimeringsproblem fås när alla funktionerna $f(\mathbf{x})$ och bivillkoren $c_i(\mathbf{x})$ i (2.1)-(2.3) är linjära funktioner av $\mathbf{x}$ [3]. Detta resulterar i ett linjärt programmeringsproblem, där standardformen är enligt (8.1.1) i [3].

$$\underset{\mathbf{x}}{\text{Minimera}} \quad f(\mathbf{x}) \triangleq \mathbf{c^T x} \tag{2.4}$$

$$\text{Givet att} \quad \mathbf{Ax} = \mathbf{b}, \qquad \mathbf{x} \geq \mathbf{0}, \tag{2.5}$$

där $\mathbf{A}$ är en m×n matris, och $m \leq n$. De flesta enkla linjära problem kan anpassas till denna standardform utan större svårigheter.

Medan flera programmeringsspråk tillhandahåller applikationer och bibliotek för att lösa linjära programmeringsproblem så är Python det mest använda [4]. Detta då Python är ett väl använt och dokumenterat programmeringsspråk, vilket gör det lätt för användare att arbeta med [4]. Python har en stor mängd bibliotek avsedda för optimering och linjär programmering [4]. Ett av dessa bibliotek är Python Optimization Modeling Objects, förkortat Pyomo [4].

Pyomo är ett bibliotek som specifikt utvecklats för optimering [4]. Det används för att definiera optimeringsmodeller, skapa konkreta problem, och lösa dessa med hjälp av standard lösningar [4]. En av dessa standard lösningar är GNU linear programming kit, förkortat GLPK. GLPK löser optimeringsproblem genom att ta det indata som är givet i dess inbyggda funktioner Constraint och Objective som analyserar dess struktur, för att sedan välja en passande algoritm baserat på problemets storlek och komplexitet [5]. För linjär programmering använder GLPK simplexmetoden, där den då rör sig längs gränserna av den möjliga regionen för att hitta den optimala lösningen på problemet [5]. Den gör denna process iterativt, och itererar då mellan bivillkor och variabler för att hitta det optimala resultatet [5]. I detta sammanhang syftar en variabel på en symbolisk representation av ett värde som då kan ändras under programmets gång, det används för att manipulera och lagra data. Parametrar syftar på de värden som skickas till en funktion eller en metod när den anropas, för att användas under dess exekvering.

2.2 Mätning av solprognosernas noggrannhet

Vid prognoser och uppskattningar av data så spelar mätningens noggrannhet en stor roll. MAPE, *Mean Absolute Percentage Error* används oftast till prognosmodeller då den mäter absolutbeloppskillnaden mellan det uppskattade och det faktiska värdet vid varje tidpunkt [6]. Dessa summeras sedan och divideras med antalet tidspunkter för att få MAPE i form av procent. Detta redovisas enligt följande ekvation som hämtades ifrån [7].

$$\text{MAPE} = 100\frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| \tag{2.6}$$

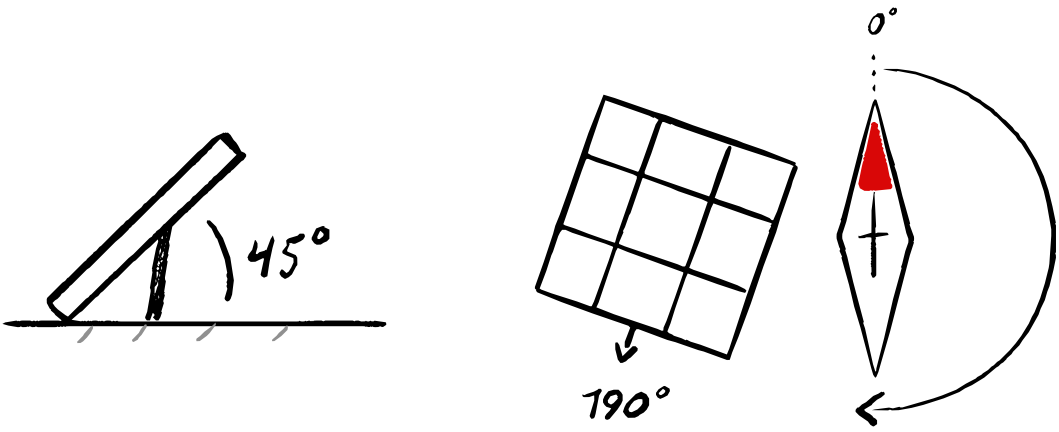
Trots att MAPE metoden är ett vanligt tillvägagångsätt för att mäta prognosers noggrannhet så har den sina begränsningar. När antalet värden i procentberäkningen är liten eller närmar sig noll leder det till extremt stora resultat, vilket gör MAPE meningslöst [6]. Även över- och underskattnings uppskattningar kan leda till stora fel vid MAPE beräkningar då de behandlas likadant [6]. I vissa situationer kan fel i prognoser ha olika konsekvenser beroende på om de är över- eller underskattade [6]. En sådan situation kan vara energiprognoser [6].

En alternativ metod för noggrannhetsmätning är *Normalized mean absolute error* som beräknas genom att ta medelvärdet av de absoluta felen och normalisera med skillnaden mellan maxvärdet och minvärdet för den verkliga datamängden [6]. NMAE metoden överkommer de begränsningar som MAPE har vid datavärden som närmar sig noll eller är negativa. NMAE normaliserar felet genom att dividera det med området för de faktiska värdena vilket ger ett mer balanserat mått på noggrannheten. Vid prognoser för att förutsäga efterfrågan på elbelastningen kan ibland solproduktionen enligt prognosen närma sig noll, vilket gör NMAE till en mer lämplig mätmetod vid den typen av prognoser [6].

2.3 Forskningsanläggningen på Björkö

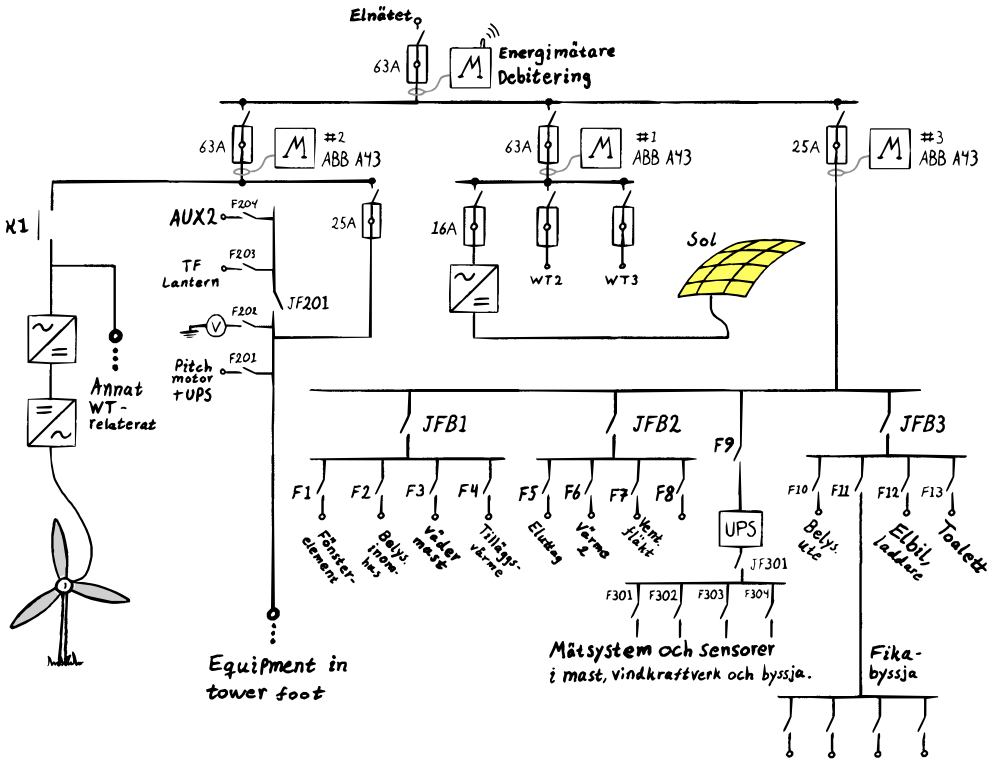
I den högra byggnaden, av de två som visas i Figur 2, ämnar projektgruppen implementera kontrollsystemet och tillsätta den hårdvara som krävs för att tillåta kommunikation, styrning och datainsamling. Den huserar ett kontor, mätutrustning, styrsystem samt kraftelektronik för vindkraftverket och solpanelerna på dess tak. Den benämnas härefter som “byggnaden”.

Enligt skriftlig korrespondens med anläggningens ansvarige, Magnus Ellsén, är taket bestyckat med 15 solpaneler av typen *Pallas 230 SM3 PBB*, med en maximal effekt på 230 W per panel. Dessa angavs ha en lutning på 45° relativt marken, och står 190° i azimut (rotation relativt nordlig riktning). Panelernas riktning visas i Figur 3.



Figur 3: En illustration av solpanelernas lutning och azimut.

Byggnaden har ett flertal redan befintliga mät- och datainsamlingsinstrument. Anläggningen är ansluten till Ellevio, som är en stor elnätsleverantör i Västra Götaland [8]. Vid diskussion med Magnus Ellsén under besök på Björkö, blev gruppen informerade om att Ellevio ger kunder möjlighet att se och följa importerad och exporterad energi på företagets hemsida. De sparar dessutom historisk data för kunder, så det går att hämta och analysera historisk data för import och export.



Figur 4: Det elektriska kopplingsschemat för utrustningen i anläggningen på Björkö. Alla elmätare är markerade med M, och # anger dess konfigurerade Modbus-adress. Elmätare 1 hanterar anläggningens solpaneler, mätare 2 är placerad framför vindkraftverkets utrustning och mätare 3 är placerad framför all passiv förbrukning i byggnaderna samt dess olika mätsystem. Det finns ytterligare utrustning bakom $K1$ som inte visas, då vindkraftverket inte var aktuellt i detta projektet. Ritningen är baserad på en ritning av Magnus Ellsén.

Vidare så finns det tre elmätare och en temperaturmätare på byggnaden. Elmätarna är kopplade enligt figur 4. Temperaturmätaren var en enkel digital termometer, med en inbyggd skärm för att visa temperatur. Tyvärr erbjuder den inget sätt att lagra eller hämta temperaturen utöver visuell inspektion, vilket konstaterades vid gruppens första besök ute vid byggnaden. Elmätarna är av modell A43 och är tillverkade av ABB. De kan kommunicera med protokollet Modbus över RS-485 [9].

Byggnadens befintliga uppvärmning hanteras av ett elektriskt element under kontorsfönstret, som är direktkopplat på en av faserna bakom den tredje elmätaren. På grund av direktkopplingen är det tämligen svårt att styra uppvärmningen i byggnaden. I byggnaden fanns också en omriktare som kördes när solcellerna producerade el, och i det fallet kan den alstra en del värme. I kommunikation med labbansvarig Magnus Ellsén framgick det också att när vindkraftverket bromsas görs det med ett effektmotstånd i byggnaden som omvandlade energin till värme.

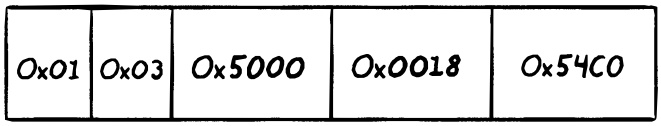
2.4 Kommunikation med elmätare

För att hämta mätvärden från byggnadens tre elmätare fanns det en del olika kommunikationsprotokoll att välja mellan. Protokollen som användes i detta projektet beskrivs i de följande två underavsnitten.

2.4.1 Modbus

Elmätarna kan kommunicera genom ett protokoll som heter Modbus, ett buss-protokoll som [9] beskriver i detalj. Bussen består av tre kablar kopplade enligt standarden RS-485, där alla enheter är anslutna till samma tre kablar. Varje ansluten enhet har en unik adress, och tillhandahåller tjänster på denna buss i form av funktionskoder. Data läses och skrivs med dessa koder till och från 16-bitars register. I elmätarnas fall finns alla mätvärden sparade som en eller flera register.

För att läsa data från elmätarna, skickar en huvudenhet en förfrågan på bussen, som specificerar den önskade enhetens adress, en funktionskod, ytterligare data till funktionen, och slutligen en kontrollsumma på 16 bitar. Den adresserade enheten svarar sedan med en liknande struktur. Ett exempel på en sådan förfrågan visas i Figur 5 här nedan.



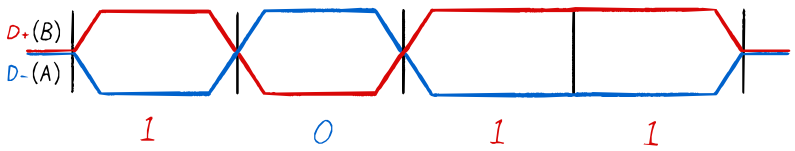
Figur 5: Ett exempel på ett Modbus-paket. Data är angiven på hexadecimal form. En siffra täcker 4 bitar, så två siffror motsvarar en oktett (8 bitar). Denna förfrågan är adresserad till enhet 1, har funktionskod 3 för "Read holding registers", säger att startadressen är 0x5000 och att 24 register önskas läsas. De sista 2 oktetterna är en kontrollsiffra.

2.4.2 RS-485

Elmätarna kommunicerar via Modbus över RS-485, som nämndes tidigare. RS-485 är en elektrisk standard, som säger hur det fysiska lagret av kommunikationen ser ut, hur det kopplas, vilka spänningar som motsvarar ett och noll, och så vidare [10].

Signalen på bussen är differentiell, vilket innebär att det är skillnaden mellan två spänningar som avgör det logiska värdet som skickas [10]. Hur information skickas differentiellt illustreras av Figur 6. Det är av fördel att den signalkabel som används för denna typen av kommunikation är ett tvistat par, för då upplever bägge kablar samma elektriska brus, som sedan subtraheras bort i mottagaren då skillnaden beräknas. Med enbart ett signalpar, blir kommunikationen *half-duplex*, vilket innebär att bara en enhet kan skicka data åt gången.

I och med att en bit skickas som en skillnad mellan två signaler i ett kabelpar, går det alltså inte att skicka två bitar samtidigt (*parallellt*) i tid. Om ett längre bitmönster ska skickas kommer alla bitar i *serie* med varandra, vilket gör kommunikationen *seriell*.



Figur 6: Ett exempel på en differentiell signal, där den vertikala axeln är spänning. D_+ och D_- är positiv respektive negativ signal. En logisk etta skickas då $D_+ > D_-$, och en logisk nolla skickas då $D_+ < D_-$. A och B är motsvarande benämningar som användas av exempelvis databladet för MAX483 [11].

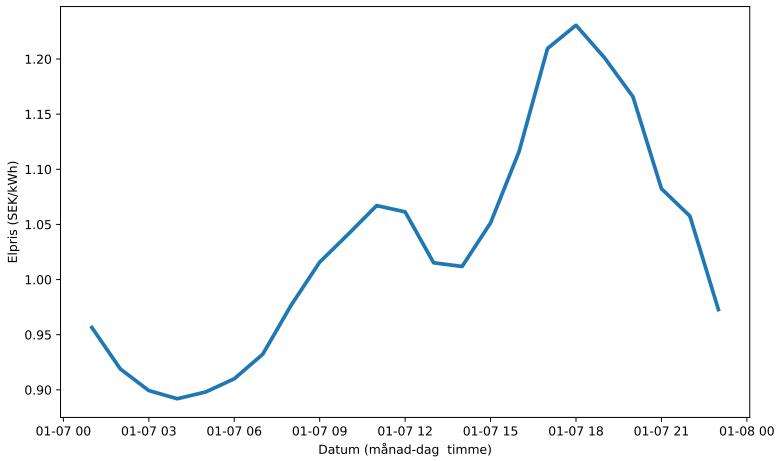
2.5 MQTT

MQTT är ett så kallat publicera/prenumerera-protokoll, med vilket *klienter* eller *servrar* skickar meddelanden [12]. Protokollet fungerar mycket likt hur information flödar genom en tidningsredaktion. Redaktionen har källor som skickar in ny information, som sedan behandlas och färdigställs till en tidning. Tidningens prenumeranter får sedan var sin kopia av tidningen, som trycks upp i det antal upplagor som behövs så att varje prenumerant får tidningen. En *server* i MQTT motsvarar tidningsredaktionen, och *klienter* kan ta rollen som prenumerant eller källa.

Meddelanden för prenumeration och publikation i MQTT kategoriseras enligt så kallade *ämnen* [12]. Publicering av ett meddelande sker till ett specifikt *ämne*. När en *klient* ska prenumerera på meddelanden, måste den specificera ett *ämne*. Alla meddelanden som publiceras till en *server* med ett specifikt *ämne* kommer skickas till samtliga *klienter* som prenumererar på det *ämnet*.

2.6 Elpris

Kostnaden för el bestäms av många komplexa faktorer [13]. Den viktigaste delen för elpriset är det elhandelsavtal som finns för en viss byggnad [14]. Elhandelsavtalet bestämmer vad en viss mängd el kostar. Som exempel: om en ugn körs en timme med en effekt på 0.5 kW drar den 0.5 kWh (ett mått på energi, vilket kostanden baseras på). Kostnaden för den användna energin baseras sedan på elhandelsavtalet, där avtalet på Björkö är ett timprisavtal. Det betyder att priset på energi direkt följer spotpriset och därför kan variera kraftigt över en dag.



Figur 7: Spotpriserna under den 7 januari, hämtade från projektets databas.

Spotpriset är priset på elektrisk energi som upphandlats på nordpoolsbörsen och är satt per timme [15]. Eftersom det handlas på en marknad är elpriset ett resultat av efterfrågan av el och hur mycket som produceras [13]. Spotpriset en viss timme beror alltså på kostnaden för det dyraste energislaget som används för att producera den sista kilowattimmens el [13].

Elpriset beror alltså på hur mycket det kostar att producera energi, och att det kostar olika mycket att producera samma mängd energi med olika former av kraftslag [13]. Generellt sett är de dyrast att producera el med fossila bränslen, och billigast att producera med förnybar energi [13]. Flera av de förnybara energislagen (sol, vind, vågkraft) går inte att styra, utan producerar el beroende på lokala förhållanden [16]. Förhållandena går dock att modellera en kort tid i förväg, så att deras produktion kan räknas in på spotpriset en dag i förväg [15]. I Sverige är det vindkraft som står för den stora delen av variabel produktion [16].

När en enskild byggnad köper in energi betalas också en fast avgift på 53,5 öre (inklusive moms) på varje kWh [17]. Energiskatten sätts inte av elnätsföretagen utan av regeringen och adderas till spotpriset [17].

Genom att använda elen när spotpriset är billigt sparas det inte bara på kostnader utan även på miljön. Att schemalägga vissa elektriska laster för att sedan använda de när elen är billig är vad som benämns flexibilitet.

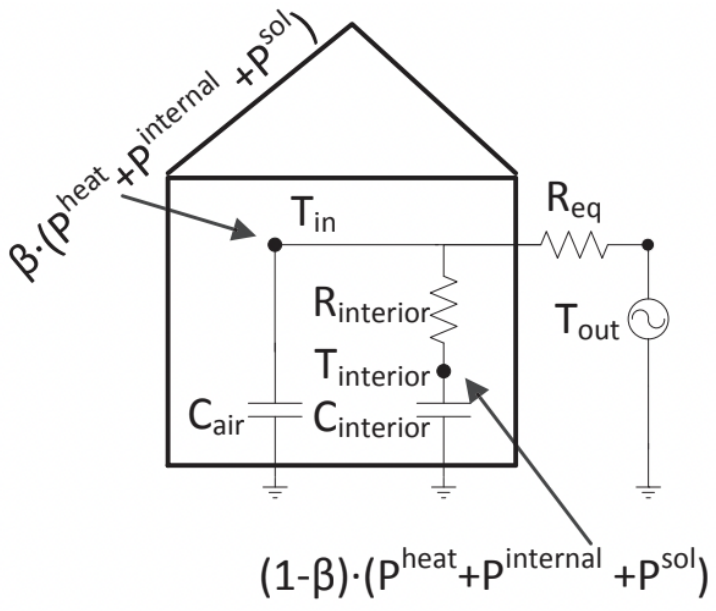
2.7 Termisk byggnadsmodell

För att veta hur mycket el det går åt att värma upp en byggnad behövs en termisk modell. En termisk modell är en representation av de energiöverföringar och temperaturförändringar som sker i en byggnad [18]. Den termiska modellen tar hänsyn till isolation, värmetillförsel, värmeledning och värmekapacitet för att kunna beräkna temperaturens förändring över tid. En byggnads termiska egenskaper skiljer sig åt från byggnad till byggnad beroende på faktorer som volym, isolering, fönster och värmetillförsel [18]. Därav finns det många olika sätt applicera en termisk modell på en byggnad.

2.7.1 Termisk byggnadsmodell med klumpparametrar

En vanlig metod vid simulering av en byggnads termiska egenskaper är att använda sig av "Klumpparameter modellen" med en analogi i form av ett kretsschema. I denna analogi motsvarar den elektriska kapacitansen förmågan byggnaden har att lagra termisk energi och spänningen (framförallt spänningsskillnader) motsvarar temperaturen [18]. Värmeflödet är ekvivalent med elektrisk ström och de elektriska motstånden motsvarar den termiska resistansen [18]. Den resulterande temperaturen är klumpsumman av varje temperaturnod [18].

En byggnads termiska modell kan konstrueras olika beroende på dess grad av komplexitet. Den enklaste graden av en termisk modell är 1R1C, vilket betyder att den består av en resistans och en kapacitans [19]. För byggnader med isolering av högre grad krävs det termiska modeller med högre grad vilket även ökar modellens noggrannhet [19]. Med ett färdigt kretsschema kan en ekvation erhållas som beskiver hur temperaturen förändras över tid [20].



Figur 8: Bild på hur ett ekvivalent kretsschema över en termisk byggnadsmodell skulle kunna se ut [21]. Modellen innehåller 2 resistanser o 2 kapacitans vilket benämns som en 2R2C modell.

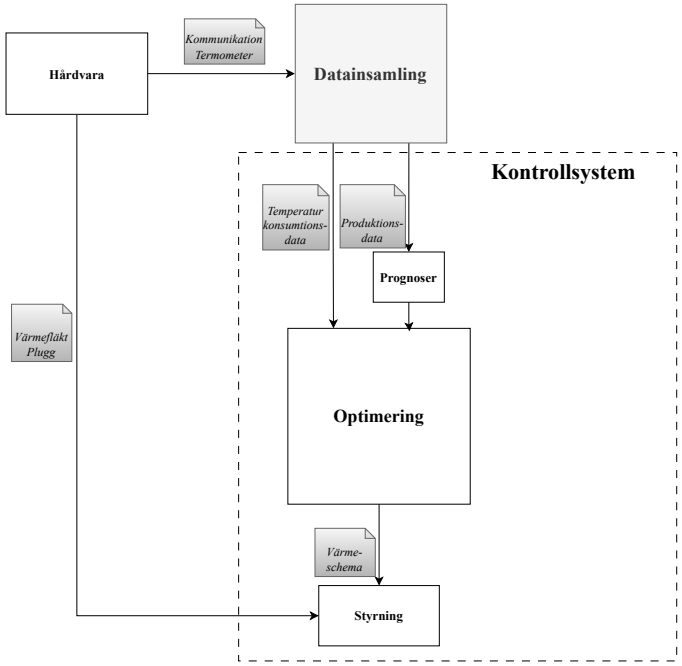
$$T_{in}[t] = T_{in}[t-1] - \frac{T_{in}[t-1] - T_{out}[t]}{R_{eq} \cdot C_{air}} - \frac{T_{in}[t-1] - T_{interior}[t-1]}{R_{interior} \cdot C_{air}} + \frac{(P_{heat}[t-1] \cdot + P_{internal} + + P_{sol}) \cdot \beta}{C_{air}} \tag{2.7}$$

$$T_{interior}[t] = T_{interior}[t-1] - \frac{T_{interior}[t-1] - T_{in}[t-1]}{R_{interior} \cdot C_{interior}} + \frac{(P_{heat}[t-1] \cdot COP + P_{internal} + P_{sol}) \cdot (1-\beta)}{C_{interior}} \tag{2.8}$$

Båda ekvationerna kommer från [21] och andvänds för att beskriva temperaturen inne i byggnaden, $T_{in}(t)$ respektive temperaturen av byggnadens innermaterial $T_{interior}$ som funktioner av temperaturen vid föregående tidsintervall $T_{in}[t-1]$, temperaturen utomhus $T_{out(t)}$, temperaturen för inredningsmaterialet $T_{interior}$ samt värmetillförseln $P_{internal}(t)$. Konstanterna som påverkar $T_{in}(t)$ är det allmänna värmemotståndet för byggnaden R_{eq} vilket inkluderar värmeledning och ventilationsförluster, C_{air} den värmeenergi som lagras i luften, $R_{interior}$ beskriver hastigheten av värmeflödet mellan fasta material och luften [21]. P_X är energitillförsel under den givna tidsperioden från de olika värmekällorna [21]. β är en viktkonstant(0-1) som används för att modellera den andel värme som hamnar i luften respektive fasta material [21]. En hög konstant motsvarar att majoriteten av den energi som tillförs byggnaden hamnar direkt i luften istället för fast material.

3 Metod

I detta avsnittet beskrivs tillvägagångssättet för projektet. Figur 9 visar de huvudsakliga momenten.



Figur 9: Metod flödesschema.

Notera att detta inte beskriver den till tiden kronologiska ordningen utan redogör för strukturen av arbetet och kopplingen mellan moment. Exempelvis innefattar datainsamlingen mer än temperatur, konsumtion och produktionsdata, men dessa anses som kritiska moment till optimeringen samt prognoser och följer naturligt från datainsamlingens beroende på implementerad hårdvara. Liknande kan sägas om de övriga stegen.

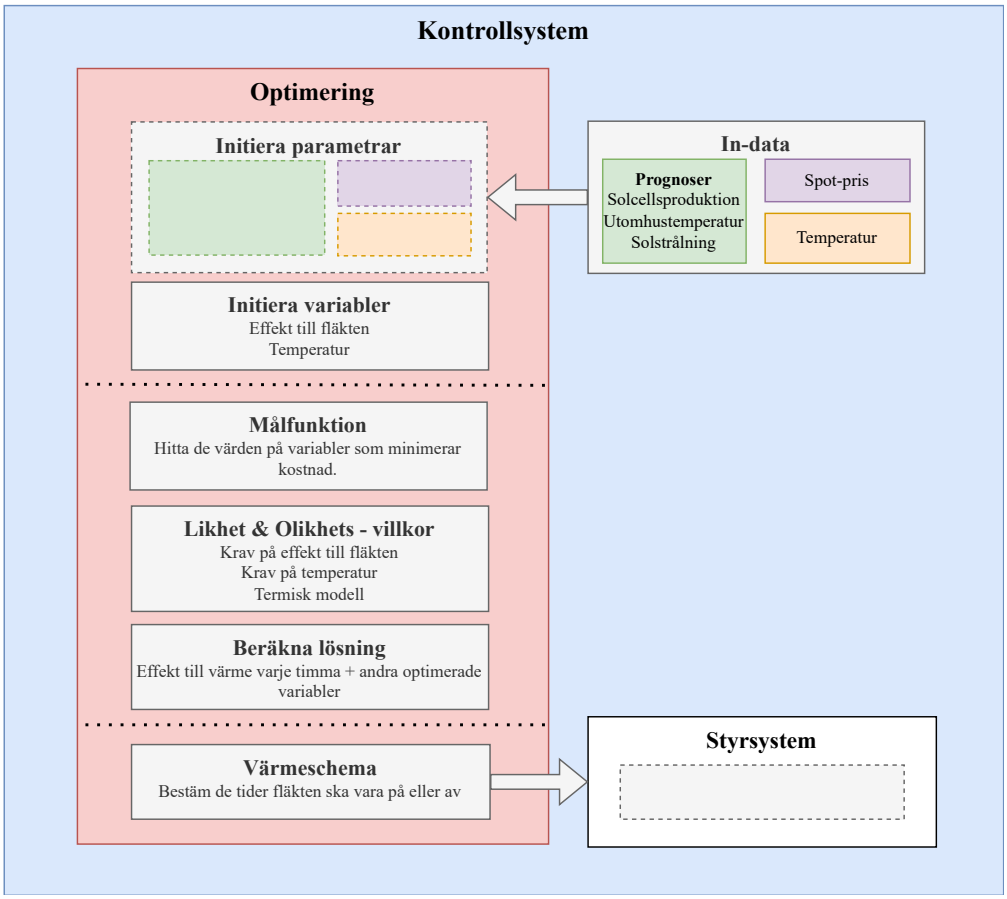
Som detta avsnitt är strukturerat behandlas optimeringen först. Eftersom val av optimering och de matematiska samband som användes avgör vilken data/prognoser som ska användas är detta lämpligt. Från Avsnitt 2.7.1 var det exempelvis tydligt ur (2.7) hur utomhustemperaturen kunde användas. Genom att förstå modellen projektet designats runt blir motiveringar för delmomenten tydligare.

Samtliga prognoser omfattar solproduktion och utomhustemperatur. För solproduktionen användes plattformen Rebase [22]. Utomhustemperaturen och elpriset hämtades från SMHI respektive Entso-e med hjälp av deras applikationsgränssnitt [23][24].

Datainsamling behandlar huvudsakligen lösningar för kommunikation och insamlande av data från energi-mätarna, vilket följer strukturen i metoden. Elpriset och prognosen för utomhustemperaturen kan också behandlas som datainsamling, men faller under kontrollsystemet i metoden för deras nära anknytning till optimeringen.

Styrsystemet, som likt datainsamlingen krävde hårdvarulösningar, är den metod med vilken kontrollsystemet faktiskt styr uppvärmningen. Som går att se i Figur 9 användes till projektet en värmefläkt. Anledning och detaljer beskrivs i Avsnitt 3.3.1.

3.1 Optimering



Figur 10: Blockschema för kontrollsystemet.

3.1.1 Optimeringsmodell för styrning av byggnadens värmebehov

Som nämnt tidigare resulterar de optimeringsproblem som kan beskrivas enligt standardformen i linjära programmeringsproblem. För att lösa ett linjärt optimeringsproblem krävs en definierad målfunktion, som följs av bivillkor som sätter begränsningar på målfunktionen. Då syftet med optimeringssystemet var att minimera energikostnaderna så skapades följande målfunktion:

$$\sum_{h=0}^{24} Price[h] \cdot P_{exchange}[h] \tag{3.1}$$

Målfunktionen tar alltså summan av produkten mellan $P_{exchange}[h]$ och $Price[h]$ med mål att hitta de variabler som minimerar energikostnaden.

Parametrar, som tidigare nämnts, refererar till de värden som skickas till en funktion vid anrop. För att optimera processen valdes följande parametrar: solprognos, utomhustemperatur och spotpris. Solprognosen hämtas från Rebase och beskrivs närmare i Avsnitt 3.1.3. Denna prognos

återfinns i parametern $P_{production}$. Utomhustemperaturen, representerad av parametern T_{out}, hämtas från SMHI, vilket förklaras utförligt i Avsnitt 3.1.4. Spotpriset på el, betecknat som $Price$, hämtas från ENTSO-E och beskrivs i Avsnitt 3.1.5. Samtliga definierade på timintervall med värden från 00:00 till 23:00.

Fortsättningsvis valdes variablerna, som tidigare nämnt definieras dessa som en symbolisk representation av ett värde som kan ändras under programmets gång. T_{in} för temperaturen inomhus i anläggningen och $T_{interior}$ som motsvarar temperaturen för inredningsmaterialet. P_{heat} för energin till värmefläkten och slutligen $P_{exchange}$ för den energi som ska importeras eller exporteras.

De bivillkor som begränsar målfunktionen specificeras enligt följande ekvationer, med definition av domänerna hos variablerna först:

$$T_{in}, T_{interior}, P_{exchange}[h] \in \mathbb{R} \tag{3.2}$$

$$P_{heat} \in \mathbb{R}^{+} \tag{3.3}$$

Sedan följer villkoren för temperaturerna den första timmen, (3.4) och (3.5) samt temperaturernas gränser. Notera att (2.7) och (2.8) också är bivillkor:

$$T_{in}[h] = T_{in\_initial}, \quad \text{Givet att } h = 0 \tag{3.4}$$

$$T_{interior}[h] = T_{interoir\_initial}, \quad \text{Givet att } h = 0 \tag{3.5}$$

$$T_{\min} \leq T_{in}[h] \leq T_{\max} \tag{3.6}$$

Sist behandlas kopplingen mellan solproduktionen och uppvärmningen samt ett övre tak på $P_{heat}[h]$.

$$P_{heat}[h] = P_{exchange}[h] + P_{production} \tag{3.7}$$

$$0 \leq P_{heat}[h] \leq 2 \tag{3.8}$$

Enligt (2.7) och (2.8) är värmetillförseln P_{heat} i kWh, men eftersom modellen är timbaserad går den, tillsammans med $P_{exchange}$ och $P_{production}$, under benämningen P, som normalt används till effekter. Det går alltså att tolka resultaten som effekter eller energi eftersom kW över en timma är det samma som kWh. I testerna har de behandlats som kWh. Senare i avsnittet förklaras hur optimeringens resultat hanteras.

T_{in} och $T_{interior}$ betecknar värmen i anläggningen vid en specifik timme under dygnet, och $T_{in\_initial}$ samt $T_{interior\_initial}$ betecknar den initierade temperaturen i början av optimeringsprocessen. Dessa villkor används enbart när optimeringsmodellen initieras, det vill säga när timmen är noll. Annars får både T_{in} och $T_{interior}$ sina värden från de termiska modellerna, enligt (2.7) och (2.8).

P_{heat} betecknar den totala värmeproduktionen för anläggningen varje timme, vilket är summan av den utbytta värmen för timmen, $P_{exhange}$, och den producerade värmen för timmen, $P_{production}$. P_{heat} ryms enbart mellan ett intervall från 0 till 2 kW, då värmefläkten som är installerad ute i anläggningen var inställd på 2 kW.

Vidare så bestämdes temperaturintervallet till 20 °C och 25 °C, efter kontakt med Magnus Ellsén via email. Det som kan skada utrustningen i byggnaden är frost och överhettning. Det sker enbart vid extremt låga och höga temperaturer, och intervallen sattes därför utefter bekvämlighet för

de som rör sig inne i byggnaden. Alltså sätts temperaturens nedre gräns, T_{min}, till 20 °C och temperaturens övre gräns, T_{max}, till 25 °C efter förslag från Magnus Ellsén.

För att lösa optimeringsproblemet används Pythonbiblioteket Pyomo. Pyomo är som nämnt tidigare ett bibliotek som är skapat specifikt för optimering, och förenklar optimeringsprocessen genom att ha färdiga funktioner för att initiera modellen, målfunktionen och bivillkoren. Genom att initiera modellen med funktionen `ConcreteModel` startades optimeringen, medan målfunktionen i (3.1) och bivillkoren från (3.4)-(3.8) initierades med de inbyggda funktionerna `Objective` respektive `Constraint`. Då syftet med projektet var att minimera energikostnaderna så specificerades detta genom att ge minimize som input till `Objective`.

Vidare så användes GLPK som problemlösare för optimeringen, då målfunktionen och bivillkoren uppfyllde standardformen för ett linjärt programmeringsproblem, enligt (2.4)-(2.5). GLPK itererade då genom variablerna, och justerade deras värden för att förbättra målfunktionen, under förutsättning att bivillkoren som ställts i ekvationer (3.4)-(3.8) var uppfyllda. Detta resulterade i optimala värden för de variabler som initierats i målfunktionen och bivillkoren, för varje timme för det kommande dygnet.

Dessa värden användes sedan vidare för att beräkna ett värmeschema som anger vid vilka tider som fläkten på byggnaden ska sättas av och på, för att minimera energikostnaderna. Detta schemat sparas enligt en fördefinierad tidtabell.

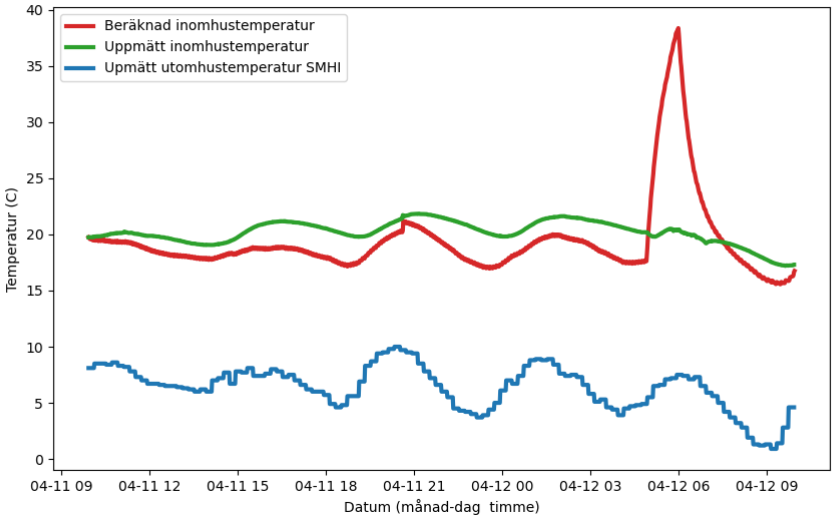
Uppdelningen gjordes så energimängden P_{heat}, för en given timma, delades med fyra för att erhålla kvartsintervall istället för timmesintervall med start i varje kvart. Tiderna avrundades till minuter.

Slutligen skapades ett enklare kontrollsystem i optimeringsmodellen. Den använder den av projektet skapade klassen `HeaterControl`, som beskrivs närmare i Avsnitt 3.2. `HeaterControl` används för att styra värmaren i anläggningen enligt tidtabellen i värmeschemat, genom att använda MQTT-protokollet för att kommunicera med den server som kontrollerar värmaren.

3.1.2 Termisk modell

Utifrån byggnadens karaktäristik anpassades en termisk modell med klumpparameter metoden enligt ordning 2R2C, se Figur 8.

Med denna modell användes (2.7) som termisk modell på byggnaden. Genom att analysera hur temperaturen förändras när värmeelementet är på/av uppskattades den passiva värmeeffekten från byggnadens utrustning. De resterande parametrarna uppskattades med grund i [21]. Därefter kunde den termiska modellen användas för att simulera byggnadens temperaturförändring över tid.



Figur 11: En representation av hur den anpassade termiska modellen förutspådde temperaturen över fyra dagar utifrån en starttemperatur och yttertemperatur som indata.

De termiska konstanterna uppskattades genom att göra en uppställning i Python av uppmätta inomhustemperaturer och utomhustemperaturer inhämtat från SMHI över 4 dagar. Då detta motsvarar den termiska indatan som det löpande systemet har bedömdes det som tillräckligt. Den första parametern som anpassades var R_{eq} då den står för hur mycket temperaturen i byggnaden beror på värmeflödet från temperaturskillnaden mellan ytter och innertemperatur.

Givet temperaturdata och inomhusstarttemperatur kan den beräknade temperaturen efter 4 dagar var samma som den faktiska. Därefter uppskattades de termiska kapacitanserna så att svängningarna inomhus följde den svängning som kom från utomhustemperaturen, samt att de liknade de svängningar som den uppmätta inomhustemperaturen följde.

På grund av tidsbrist skapades inga prognoser av solinstrållning i byggnaden, vilket medförde att P_{sol} i (2.7) sattes till 0.

3.1.3 Prognos för solcellernas produktion - Rebase

I syfte att skapa en prognos för produktionen av anläggningens solceller användes plattformen Rebase, ett företag som bland annat erbjuder AI-verktyg för att skapa energiprognoser baserat på historiska data och API-åtkomst för väderdata och prognoser [22]. I detta skaparverktyg går det att välja bland olika prognosvarianter, bland dessa väder, elförbrukning, sol och vind-produktion. För projektet valdes solproduktion. Ytterligare information om solpanelerna behövde anges i kombination med detta vilket går att observera i Tabell 1 och stämmer överens med den information som gavs i Avsnitt 2.3.

Tabell 1: Angiven data till Rebase skapar-verktyg

Latitud	*Longitud*	*Kapacitet* (kW)	*Lutning* (°)	*Azimut* (°)
57.7185	11.6838	3.468	45	190

Latitud och longitud valdes med hjälp av en integrerad karta i skapar-verktyget för att stämma överens med solpanelernas position. Kapacitet syftar till den maximala produktionsmöjligheten i kW för solpanelerna.

Fortsättningsvis har skaparverktyget avancerade inställningar bestående av fyra ytterligare valbara prognosmodeller [22]. Som plattformen fungerar går det att utöver AI-modellen välja flera prognosvarianter för sin anläggning baserade på olika ändamål. Dessa inkluderar autoregressiva, långsiktiga, väderagnostiska och fysiska prognosmodeller [22]. Samtliga valdes bort eftersom den autoregressiva modellen krävde en aktiv tillförsel av data, den långsiktiga skapade prognoser över längre perioder när systemet endast behövde se ett dygn, den väderagnostiska modellen inte tog i hänsyn historisk väderdata till modellträningen och den fysiska inte använde historiska data [22]. Utöver detta var det möjligt att välja mellan kvart, halvtimme, timme eller dag-upplösning [22]. För projektet valdes timmes-upplösning.

Eftersom det tog tid innan datainsamlingen kunde implementeras skapades först en modell med exportdata från Ellevio [25]. Datan är mellan åren 2014-2024 och för timvisa värden i kW. Eftersom detta innefattar exportdata är det en sämre approximation av produktionen, eftersom en del av den energi som produceras under aktiva soltimmar konsummeras av anläggningen. Då det rörde sig om mycket data och i brist på alternativ ansåg projektgruppen det fortfarande värdefullt att etablera en prognosmodell med denna data. Se Avsnitt 4.5.2 för resultat. När datainsamlingen hade implementerats användes produktionsdata insamlad mellan 08/04 och 15/04 för att skapa ytterligare en prognosmodell. Se Avsnitt 4.5.1 för resultat. För att komma åt respektive prognosmodeller i kontrollsystemet användes Rebase API [22].

3.1.4 Prognos för utomhustemperaturen - SMHI

För utomhustemperaturen hämtades prognoser från SMHI:s API och observationsdata från deras databas, vilket av SMHI är tillåtet för bland annat bearbetningar och kommersiella ändamål [24][26]. Prognoser existerar för de kommande tio dygnen, först för varje timma, sedan var tredje, sjätte och tolfte [24]. Dessa uppdateras sex gånger varje dygn [24]. Det ska noteras att man vid anropning till API:n specificerar de koordinater man önskar ha prognosen för medans observationsdatan, som i Avsnitt 4.5 och Avsnitt 3.1.2 har använts för att visa utomhustemperaturen under tester, måste plockas från en specificerad station.

Observationsdatan i projektet har tagits från station Vinga. SMHI för aktiva uppföljningar av träffsäkerheten hos deras prognoser och mellan april 2023 och mars 2024 rapporterades träffsäkerheten som lägst 74,6% i januari 2024 för prognoser utfärdade ungefär 3 dagar innan observation [27]. Det har även gjorts en studie gällande träffsäkerheten där det konstateras att de kortare dygnsprognoserna har stor träffsäkerhet [28]. Då anläggningens temperaturmätare saknar gränssnitt för fjärråtkomst och systemet endast behöver temperaturdata över kommande dygn bedömdes detta acceptabelt.

3.1.5 Elpris

Det aktuella spotpriset för en dag hämtas från ENTSO-E, en organisation skapad av Europeiska Unionen för att hjälpa sammanlänkningarna mellan Europas staters elnät. De har ett API som projektet använder sig av [23]. Det ger dagens och morgondagens spotpriser i de olika områdena i Sverige.

För att hämta data från ENTSO-E skrevs först en egen python-funktion för att skicka dataförfrågningar till APIet och tolka det svar som kom tillbaka. Sedan upptäcktes att det fanns ett redan skrivet programmeringsbiblotek för att tolka data, som användes efter kontroll av funktionalitet [29].

3.1.6 Uppskatta kostnader

De faktiska kostnaderna vid testerna i Avsnitt 4.5 beräknades som (3.9).

$$Cost_{tot} = \sum_{h=1}^{24} Price[h] \cdot (P_{heat}[h] - \sum_{m=1}^{60} P_{production,h}[m]) \tag{3.9}$$

Där m står för minuter. Det ska här noteras att $P_{production,h}[m]$ hänvisar till faktiskt uppmätta data under tester, och h anger serien för en given timme h. Dessa är mätta med minutintervall.

För att jämföra resultaten av kontrollsystemet i kontexten av projektets syfte togs medelvärdet av den summerade värmeenergin under dagen, beräknad av optimeringen, och slogs sedan ut för varje timma. Denna jämnt fördelade värmeenergi multiplicerades sedan med $Price[h]$ för att få total kostnad enligt (3.10).

$$Cost_{tot,even} = Price[h] \cdot \frac{\sum_{\mathrm{h}\in 24} P_{heat}[h]}{24} \tag{3.10}$$

(3.10) beräknas som om all energi importeras. Den tar alltså inte i beaktning någon egenproduktion alls.

3.2 Datainsamling

En stor del av projektet behövde också information om anläggningens elproduktion- och konsumtion, solenergin som producerades av dess solpaneler och hur mycket energi som förbrukades utöver uppvärmningsbehovet. I anläggningen satt det tre elmätare av typen ABB A43, som nämndes tidigare i Avsnitt 2.4.1, och för kommunikation med dem valdes protokollet *Modbus*, elektriskt kopplat enligt *RS-485*. För att sköta kommunikationen med elmätarna valdes enkortsdatorn *Raspberry Pi 4* (RP4), som placerades i anläggningen och kopplades upp så att den var via internet tillgänglig för administration genom ett virtuellt privat nätverk (VPN). RP4 har 40 pinnar för så kallad "general purpose I/O" (GPIO), vars digitala logik är på 3.3 V [30].

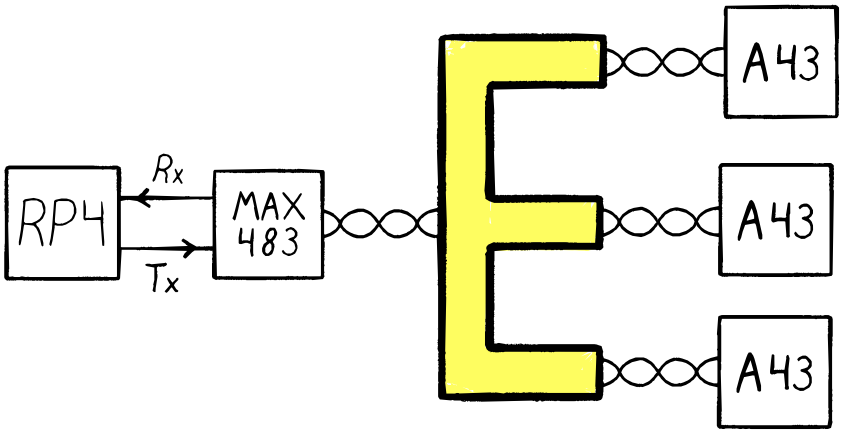
Även inomhustemperaturen behövde samlas in, vilket RP4 också fick sköta. En enkel digital termometer av modell DS18B20 användes till detta, i form av ett breakout-kort producerat av *SunFounder*. Valet var godtyckligt, baserat på vad som fanns att tillgå i komponentväg vid ett av

Chalmers maker-spaces (E.T.A.). Enligt [31] så var dess mätosäkerhet på $\pm 0.5\,^{\circ}\mathrm{C}$ inom intervallet $-10\,^{\circ}\mathrm{C}$ till $85\,^{\circ}\mathrm{C}$, vilket lämpade sig för ändamålet.

3.2.1 Kommunikation med elmätare

Logiknivån på 3.3 V som RP4 är designad för visade sig vara problematisk för kommunikation med RS-485, då signaler som skickas mellan enheter som pratar enligt RS-485 kan variera inom intervallet $-7\,\mathrm{V}$ till $12\,\mathrm{V}$ [10] (gemensam referens). Med upp till 4 gånger högre spänning gick det inte bra att bara koppla på signalkabeln på GPIO-pinnarna, för det hade riskerat att förstöra mikrokontrollern på RP4 som hanterade GPIO. Dessutom antogs det vara orimligt att kunna skicka signaler med spänningar inom intervallet $0\,\mathrm{V}$ till $3.3\,\mathrm{V}$, som av elmätarna skulle kunna tolkas på ett meningsfullt sätt.

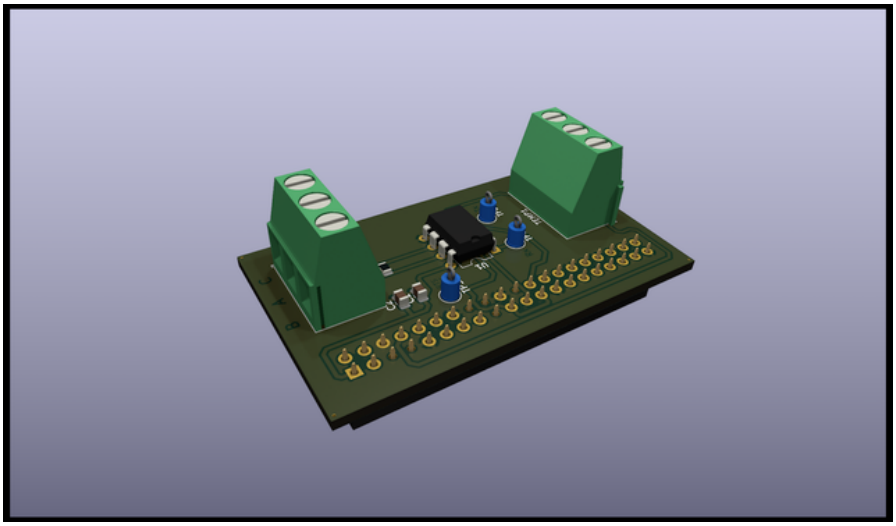
För att lösa dessa problem, designades en så kallad "hardware attached on top" (HAT). Denna HAT (läs kretskort) konstruerades runt MAX483, ett mikrochip som kan agera sändare och mottagare för RS-485, samtidigt som det, enligt databladet, också skulle kunna drivas med $3.3\,\mathrm{V}$ [11]. I effekt så använde vi MAX483 som en adapter mellan RP4 och elmätarna, som omvandlade RS-485 till spänningsnivåer kompatibla med vad enkortsdatorn klarade av att hantera. Hur detta kopplades illustreras i Figur 12. En datorgenererad illustration av det slutgiltiga kretskortet visas i Figur 13. Anslutningar till termometern konstruerades också som en del av kretskortet, i form av en av de två på kortet placerade skruvplintarna som visas i Figur 13.



Figur 12: En illustration av hur RP4 kopplades elektriskt med byggnadens 3 elmätare. De raka linjerna mellan RP4 och MAX483 motsvarar digital logik på 3.3 V. Resterande kopplingar ämnar illustrera tvistade par för kommunikation enligt RS-485.

3.2.2 Tidsseriedatabas

För att lagra data över tid, krävdes det en databas. Eftersom den typen av data som samlades in var mestadels värden som ändrades över tid, valdes *InfluxDB* för att implementera projektets databas. Det är en mjukvara som är speciellt konstruerad med just tidsserier i åtanke, som också erbjuder ett enkelt gränssnitt för att visa grafer och tabeller över insamlad data [32]. En instans av denna mjukvara installerades på en virtuell maskin som kördes på en dator hemma hos en av gruppens medlemmar. Den exponerades mot internet, så att enkortsdatorn på Björkö — men även andra med behörighet — skulle kunna skicka och läsa data till databasen, samt använda



Figur 13: Datorgenererad illustration av kretskortet. Mikrochippet i mitten är MAX483, som kopplas till några av de 40 GPIO-pinnarna på RP4 samt till den vänstra skruvplinten. Den högra skruvplinten var ett tillägg för att enkelt kunna koppla in en termometer.

dess inbyggda gränssnitt.

Upplägget för insamlingen av data till databasen var att olika program samlade information från hårdvarulösningarna, och skickade denna information till databasen över en krypterad anslutning till servern som körde *InfluxDB*. Tanken var att senare delar av projektet som behövde nyttja denna information då kunde ansluta till denna databas och läsa historiska värden, för att träna prognosmodeller eller göra analyser.

3.2.3 Mjukvara för insamling av data

Den hårdvara som hittills beskrivits har inget praktiskt syfte om det inte också finns mjukvara som nyttjar dess funktionalitet. Den HAT som skapades för Modbus-kommunikation var egentligen bara översättning mellan olika elektriska beskrivningar, men hade ingen mikroprocessor eller dylikt som kunde sköta någon kommunikation.

På RP4 installerades *Debian* som operativsystem. Två av GPIO-pinnarna som var kopplade till MAX483 konfigurerades till att vara elektriska utgångar för en seriell port i *Debian*, där pinnarna var kanaler för försändelse respektive mottagande av data. Den seriella porten representerades som en vanlig fil i operativsystemet. För att prata Modbus, kunde ett program bara läsa och skriva genom denna fil enligt dataformatet som beskrevs i Avsnitt 2.4.1, och så omvandlade MAX483 detta till differentiella signaler.

Ett problem som upptäcktes var att MAX483 krävde en signal för att växla mellan att ta emot respektive skicka data [11]. Denna egenhet var ett resultat av att RS-485 var *half-duplex* som beskrevs i Avsnitt 2.4.2. För att lösa problemet användes flödeskontroll. Två ytterligare GPIO-pinnar på RP4 användes som “request to send” (RTS) samt “clear to send” (CTS). Genom att koppla samman RTS, CTS och pinnarna på MAX483 som växlade chippets tillstånd, informerades

MAX483 om att data skulle skickas, samtidigt som förfrågan gav sig själv klartecken om att få skicka data.

För att prata Modbus med elmätarna, skrevs ett program i språket C. Programmet använde sig av ett färdigt bibliotek, *libmodbus*, som hanterade tolkning och beskrivning av data enligt Modbus paketformat [33]. De register som hämtades från elmätarna började på adresserna (hexadecimalt) 0x5B00, 0x5000 och 0x5460, där 28, 56 respektive 36 register lästes. Mer information om exakt vad som finns på dessa register går att läsa i [9], men sammanfattat så var det momentan aktiv effekt, total energi, samt total energi per fas. Denna information samlades in i interval av 5 sekunder, och skickades sedan till tidsseriedatabasen över internet.

Inomhustemperaturen samlades in på ett liknande vis. Termometern gick att avläsa genom en GPIO-pinne. Operativsystemet konfigurerades till att abstrahera kommunikationen som en fil igen, från vilken ett program kunde läsa temperaturvärden. Programmet för att läsa temperatur skrevs i språket Python, och programmerades till att samla in data varje sekund, och spara detta på samma vis som föregående program till databasen.

3.3 Kontrollsystemet

Slutligen, för att binda samman alla delar till ett fungerande kontrollsystem som reglerade uppvärmningen, behövdes det dels hårdvara för att kunna styra byggnadens uppvärmning, samt mjukvara för att omvandla optimeringsmodellens resultat till styrsignaler. Hårdvaran och mjukvaran till detta beskrivs i följande underavsnitt. Figur 14 åskådliggör hur datainsamlingen och kontrollsystemet var tänkt att hänga ihop.

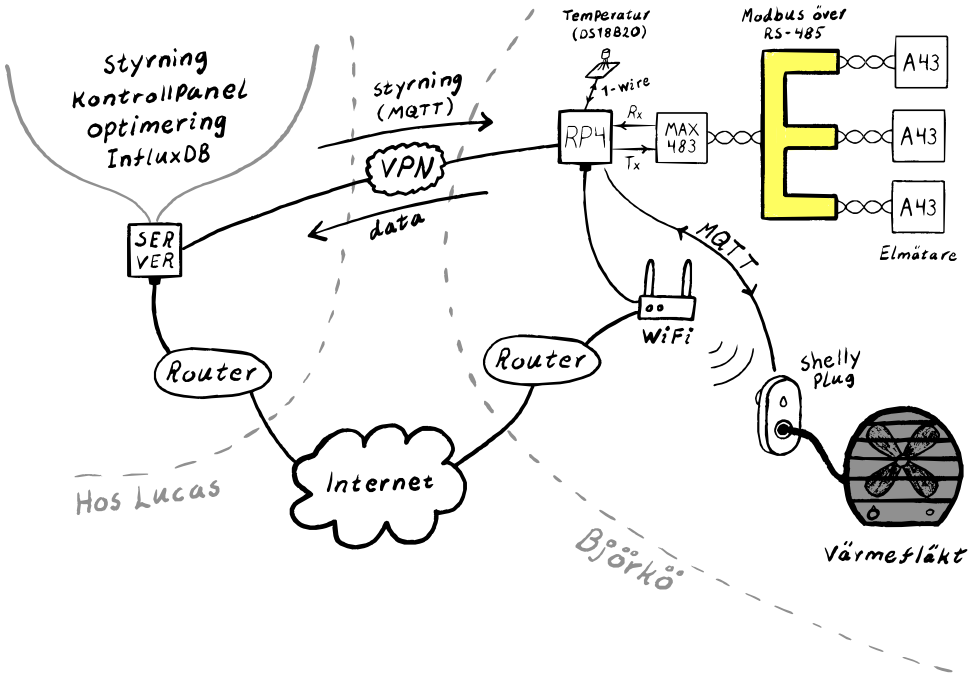
3.3.1 Hårdvara

Eftersom det visade sig vara svårt att styra elementet som fanns på plats, behövdes en ny lösning för att styra uppvärmningen. Styrbarheten implementerades med en *Shelly Smart WiFi Plug* (modell S171), en trådlöst uppkopplad väggplugg som kunde styras över den trådlösa anslutningen. Som element valdes en värmefläkt av okänt märke, som hade en maximal effekt på runt 3 kW. Fläkten ställdes in för att gå på 2 kW med en termostat på runt 30 °C efter diskussion med Magnus Ellsén vid besök på Björkö. Samtliga två enheter lånades av HSB Living Lab. Den trådlösa väggpluggen anslöts till ett ledigt vägguttag inne i kontorsdelen av huvudbyggnaden. Värmefläkten anslöts sedan till väggpluggen, och ställdes på golvet i samma utrymme, under ett skrivbord. Från det att värmefläkten installerades ombads anläggningsansvarige att ställa in fönsterelementets termostat på 10 °C, för minimal påverkan på projektets uppvärmning.

3.3.2 Mjukvara

Väggpluggen kopplades upp på byggnadens WiFi-nätverk vilket gjorde den tillgänglig för RP4 att kommunicera med. Den körde ett enkelt webb-baserat gränssnitt genom vilket inställningar kunde göras. Till kommunikation och styrning valdes protokollet MQTT, då väggpluggen hade stöd för detta [34]. MQTT beskrivs mer detaljerat i Avsnitt 2.5. På RP4 installerades *Mosquitto*, en mjukvara som implementerade en så kallad *server*. Till denna anslöts väggpluggen som en *klient*. Pluggen rapporterade tillstånd och energiförbrukning till *Mosquitto* på vissa specifika *ämnen*, och lyssnade på styrsignaler också genom *ämnen*.

Kontrollsystemets mjukvara implementerades som ett program skrivet i Python, som band ihop alla olika delar av projektet: inhämtning av prognoser, optimering och styrning. Programmet



Figur 14: Planerad nätverks- och enhetskarta över alla delar av kontrollsystemet. Datainsamling hanterades av RP4, och sparades till InfluxDB. Styrningen och dess stödmjukvaror kördes på en server hemma hos en gruppmedlem, och kommunikationen mellan servern och RP4 fördes över ett krypterat virtuellt privat nätverk. Hur hårdvaran och mjukvaran för styrningen hängde ihop beskrivs i följande underavsnitt.

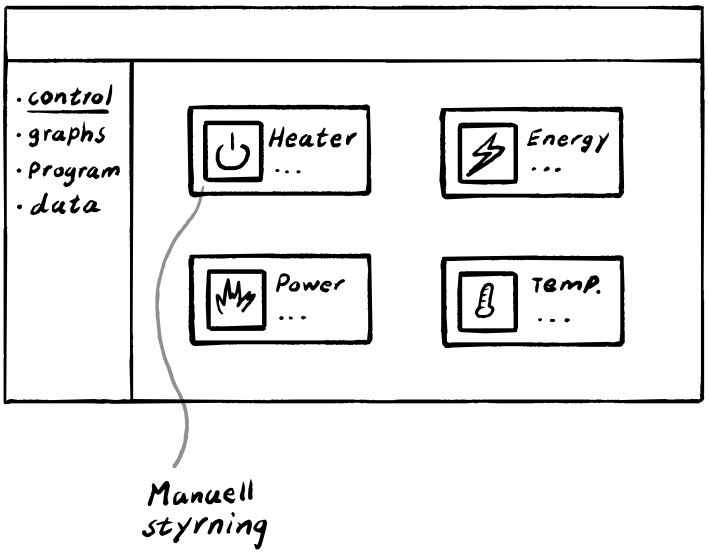
kördes på samma dator som InfluxDB var installerad på. Programmet skrevs för att optimera energikostnaden under ett dygn, så alla data inhämtades för nästkommande dag. En linjär optimering kördes sedan på kommande dags prognoser och priser, och av detta erhölls en lista med energi per timme för kommande dag. Kontrollsystemets kod finns listat som `control.py` i Appendix A, och alla mindre ingående delar för insamling av data är också listade i efterföljande appendix.

Det sista stycket av `control.py` hanterade styrningen av värmefläkten. I styrningen togs värmeschemat från optimeringen och delade upp respektive timme i hur mycket värmefläkten skulle vara av och på under den timman – värmeschemat – i form av en lista med par av datum, klockslag och tillstånd. För varje tid i listan inväntades sedan klockslaget med hjälp av funktioner från `SleepUtils.py`, för att sedan sätta på eller stänga av värmefläkten via en instans av klassen `HeaterControl`. Koden för `SleepUtils.py` och `HeaterControl.py` finns i Appendix F respektive Appendix D.

3.3.3 Kontrollpanel

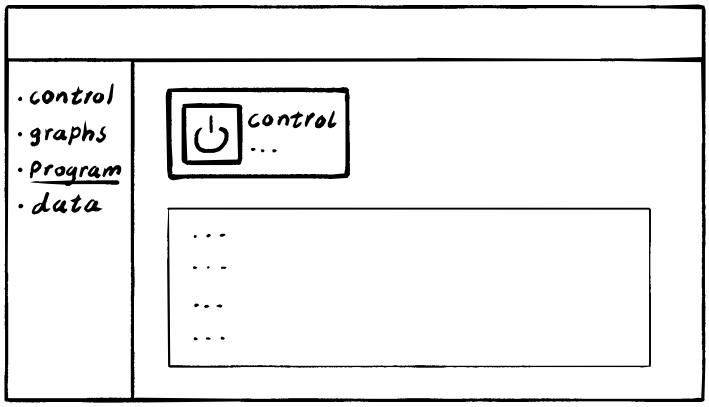
För att underlätta med att testa kontrollsystemet, och enkelt kunna styra väggpluggen manuellt utan att behöva krångla med virtuella nätverk och terminalkommandon, skrevs ett program i

språket Go som tillgängliggjorde start och stop av kontrollsystemet samt manuell styrning av väggpluggen genom ett webb-baserat gränssnitt. Programmet samlade även in information om väggpluggens tillstånd och skickade detta till InfluxDB. Hur kontrollpanelen för värmefläkten skulle se ut visas i Figur 15.



Figur 15: En skiss över hur kontrollpanelen för manuell styrning av värmefläkten var tänkt att se ut. En knapp skulle kunna stänga av och sätta på uppvärmningen, med några rutor intill för att visa effekt, temperatur och liknande.

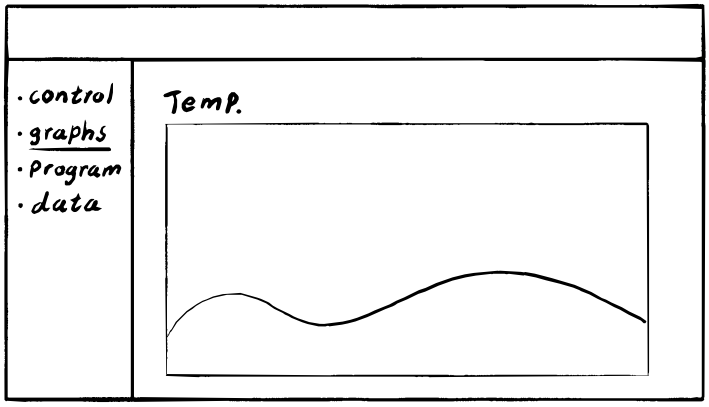
Detta program fick också som ansvar att köra kontrollsystemet som beskrevs i Avsnitt 3.3.2. All programkod för kontrollsystemet låg sparad på Github i en privat repository, och så fort en gruppmedlem uppdaterade koden på Github så laddade kontrollpanelen ner den nya versionen till en lokal mapp på samma dator. Under en flik som hette “program” kunde användare sedan köra koden för kontrollsystemet. Hur detta såg ut visas i Figur 16.



Figur 16: På panelens programflik var det tänkt att användaren skulle kunna stänga av och sätta på kontrollsystemet, samt se dess utdata i en textruta under knappen.

Tanken var även att kontrollpanelen skulle göra det möjligt för gruppens medlemmar att snabbt få översikt över systemets tillstånd med hjälp av grafer. Fliken *grafer* i kontrollpanelen var tänkt att se ut som Figur 17, med fler grafer än bara för temperatur som illustrerat. Effekt, temperatur, och elpris, samt värmefläktens tillstånd över tid var tänkt att visas på olika delar av den fliken i panelen.

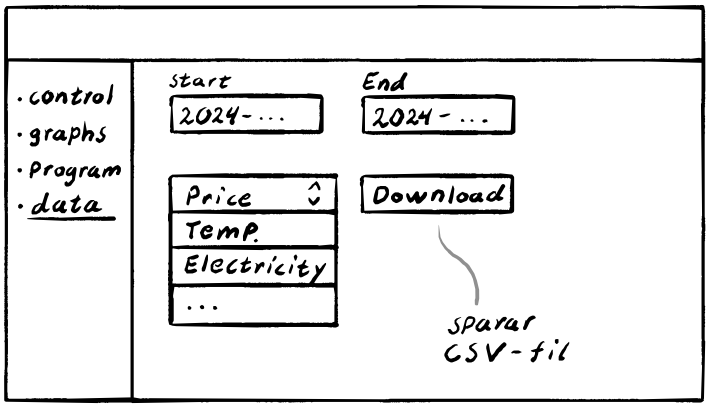
Det visade sig vara för tidskrävande att försöka programmera grafritning till kontrollpanelen, så istället användes mjukvaran *Grafana* för att visualisera data. *Grafana* är en färdig lösning för att visualisera data, som går att koppla till många olika slags databaser för att sedan rita grafer av hämtad data [35]. En så kallad "dashboard" konstruerades med data som hämtades i realtid från projektets databas, så att datainsamlingen presenterades i grafer samlade på samma sida.



Figur 17: Fliken för grafer skulle visa grafer över insamlad data som sparats till databasen. Tanken var att visuell presentation av korrelerad data skulle göra det enkelt att få en snabb uppfattning om systemets tillstånd och att kunna felsöka beslut fattade av kontrollsystemet.

För att kunna exportera data från databasen skulle en flik kallad *data* tillhandahålla enkel exportering av databasens innehåll till CSV-filer, för användning i analys och konstruktion av den termiska modellen samt optimeringsmodellen, eller för att göra någon annan form av analys. Hur denna fliken var tänkt att se ut visas i Figur 18.

Som nämndes i Avsnitt 3.2.2 så erbjöd InfluxDB ett enkelt gränssnitt för att visa grafer över insamlad data samt möjligheten att spara ner data från en serie till en CSV-fil. Den inbyggda funktionaliteten i InfluxDB visade sig vara tillräckligt för det ändamålet. Därför implementerades inte fliken *data* i kontrollpanelen.



Figur 18: För att kunna analysera data eller att göra figurer till rapporten, var det tänkt att en flik skulle tillhandahålla funktioner för export av data från databasen. På fliken skulle användare kunna välja en tidsserie samt dess start- och slutdatum, och sedan spara denna data lokalt på den egna datorn.

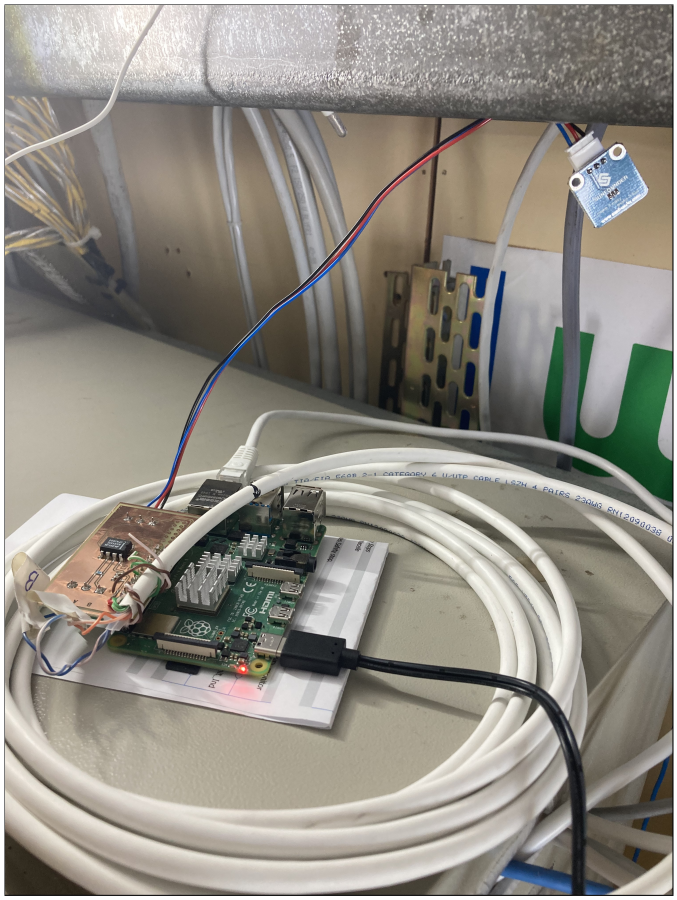
4 Resultat

I detta avsnitt presenteras det resulterande kontrollsystemet, samt den färdiga lösningen för datainsamling i byggnaden ute på Björkö. Vidare kördes det ett fåtal experiment där kontrollpanelen, optimeringen och styrningen testades. Experimentens resultat presenteras också i detta avsnitt.

4.1 Datainsamling

I figur 19 visas RP4 installerad i byggnaden på Björkö, nära takhöjd. Den HAT som designades för kommunikation med elmätarna var kopplad till elmätarna via ett tvistat kabelpar som gick längst med kabelstegen i taket bort till elmätarna. Termometern för inomhustermperatur kopplades till den andra skruvplinten som i Figur 19 är undertill på baksidan. Systemet kopplades till byggnadens nätverk via vanlig ethernet-kabel.

Kod för det program som kommunicerade med elmätarna och skickade data till InfluxDB var för stor för att inkludera som en listning i rapportens appendix, så den publicerades på Github och går att hitta i [36]. Kod för att avläsa termometern kopplad till RP4 HAT:en, och försändelse av dess data till InfluxDB, är listad i Appendix G. Programmet som sparade elpriset till InfluxDB för användning i Grafana är listad i Appendix H.



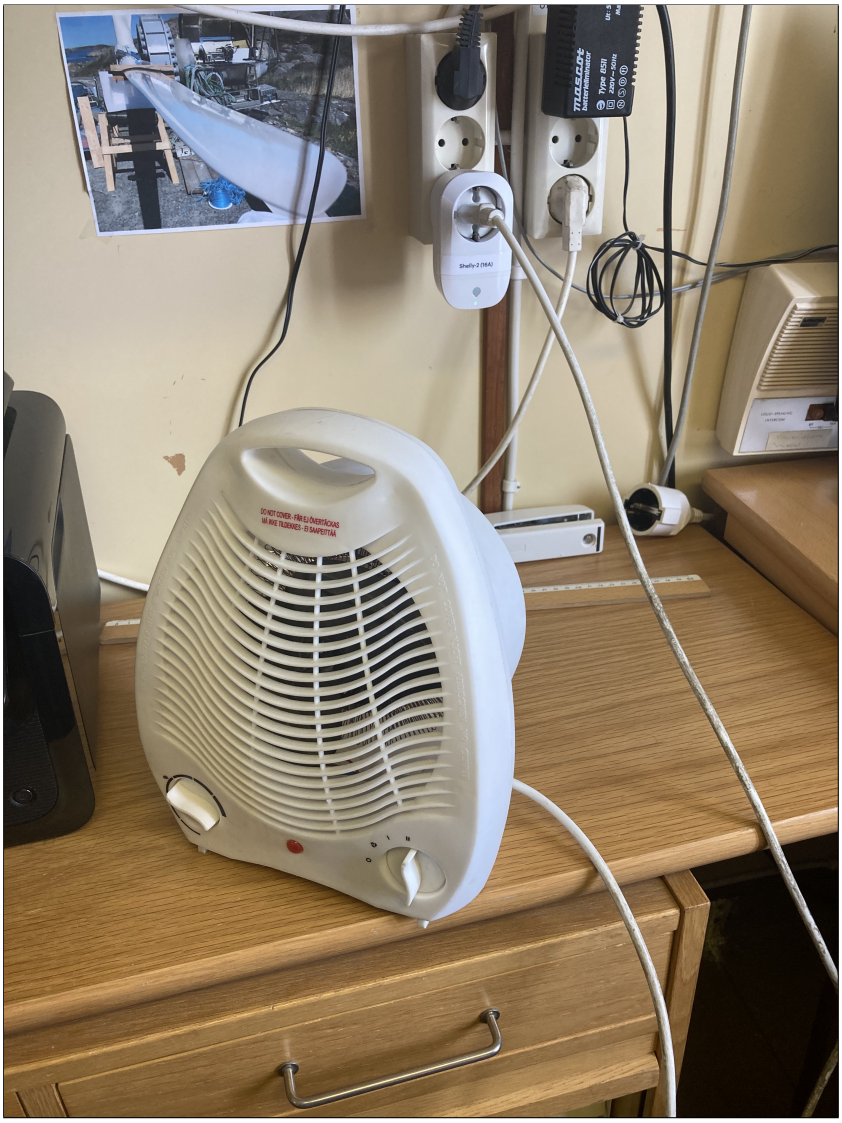
Figur 19: RP4 installerad med HAT och termometer, ute i byggnaden på Björkö. Den vita kabelhögen är vanlig ethernet-kabel med dess tvistade kabelpar exponerade i ändan. Ett av dessa par användes för Modbus över RS-485, och ett annat par användes för delad referens. Den blåa kretsen hängandes från kabelstegen är termometern, som var ansluten till HAT:en. Den vita kabeln som går in vid baksidan av RP4 är systemets uppkoppling till byggnadens nätverk, och sedan ut på internet.

4.2 Kontrollsystem

Kontrollsystemet bestod av flera olika komponenter, vars resultat presenteras i följande underavsnitt.

4.2.1 Styrbar uppvärmning

Kontrollsystemets slutsteg var styrning av uppvärmningen i byggnaden. Som beskrevs i metod användes en väggplugg för att styra en värmefläkt. En bild på värmefläkten och väggpluggen visas i Figur 20.

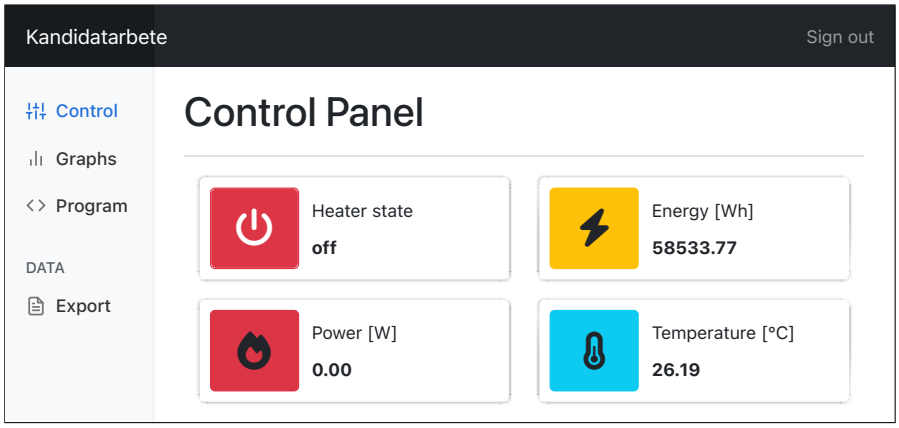


Figur 20

4.2.2 Kontrollpanelen

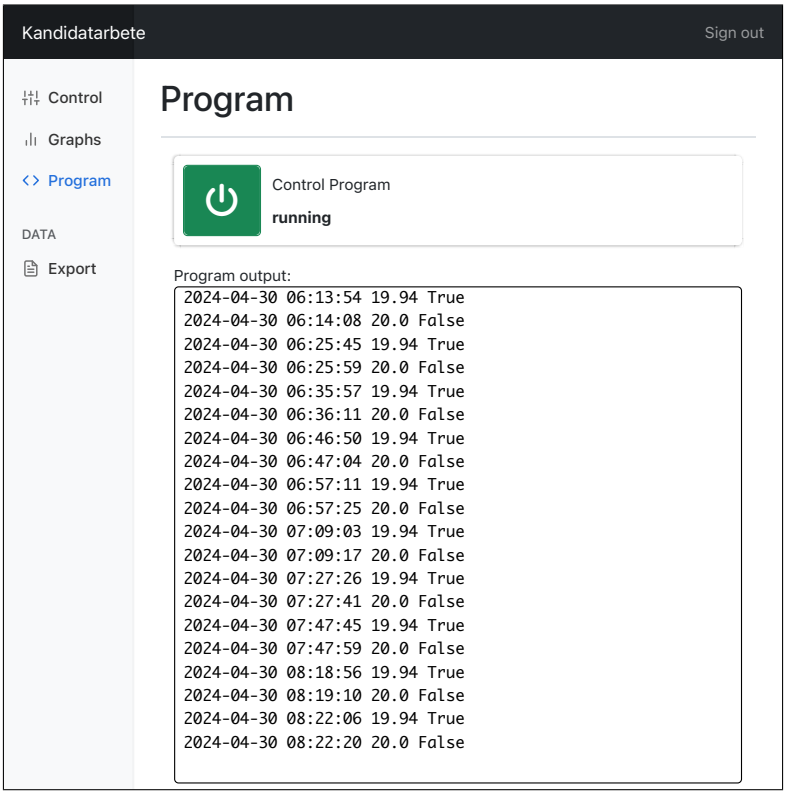
I Figur 21 visas fliken *control* som beskrevs i Avsnitt 3.3.3 och skissades i Figur 15. Vidare visas resultatet för fliken *program* i Figur 22. För att styra kontrollsystemet räckte det att implementera dessa två sidor.

Som beskrevs i Avsnitt 3.3.3, implementerades aldrig de andra två planerade flikarna i kontrollpanelen, *grafer* och *data*. De ersattes med färdiga programlösningar som implementerade de önskade funktionerna. Istället för fliken *data* användes det inbyggda gränssnittet i InfluxDB. Hur detta gränssnitt såg ut visas i Figur 24. Istället för fliken *grafer* används *Grafana*. Figur 25 visar hur den "dashboard" som konstruerades såg ut.

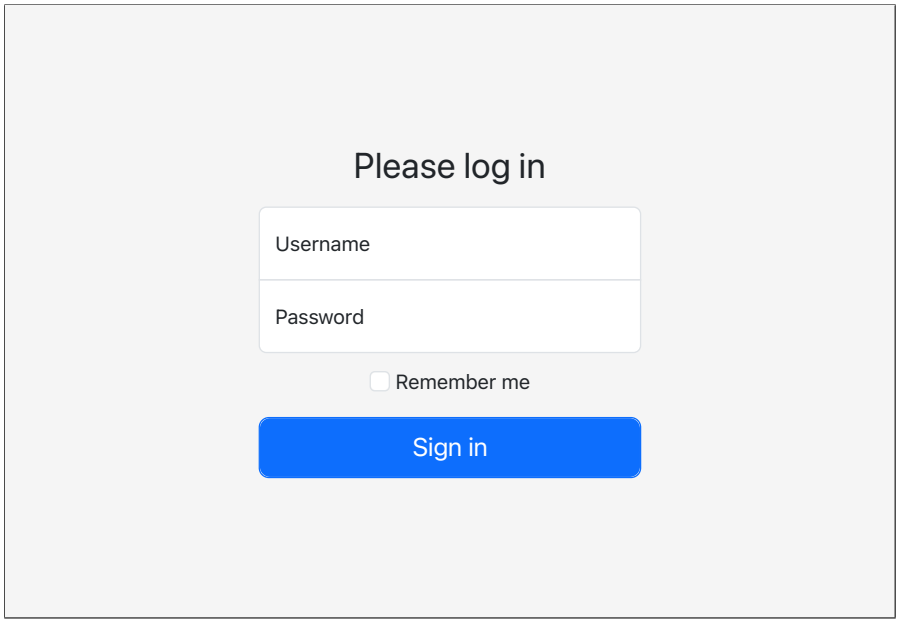


Figur 21: På kontrollpanelen visades huruvida värmefläkten var påslagen, dess momentana effekt, rummets temperatur, samt väggpluggens energiräknarvärden. Genom att klicka på den övre vänstra knappen växlades värmefläktens läge manuellt mellan av och på.

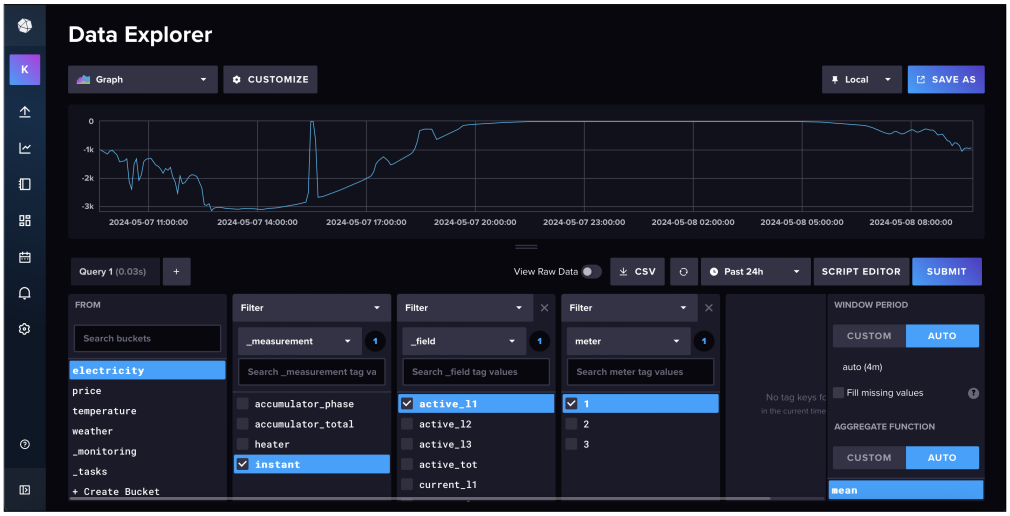
Även koden för kontrollpanelen var för stor för att inkludera som en listning i rapportens appendix, så den publicerades också på Github och går att hitta i [37].



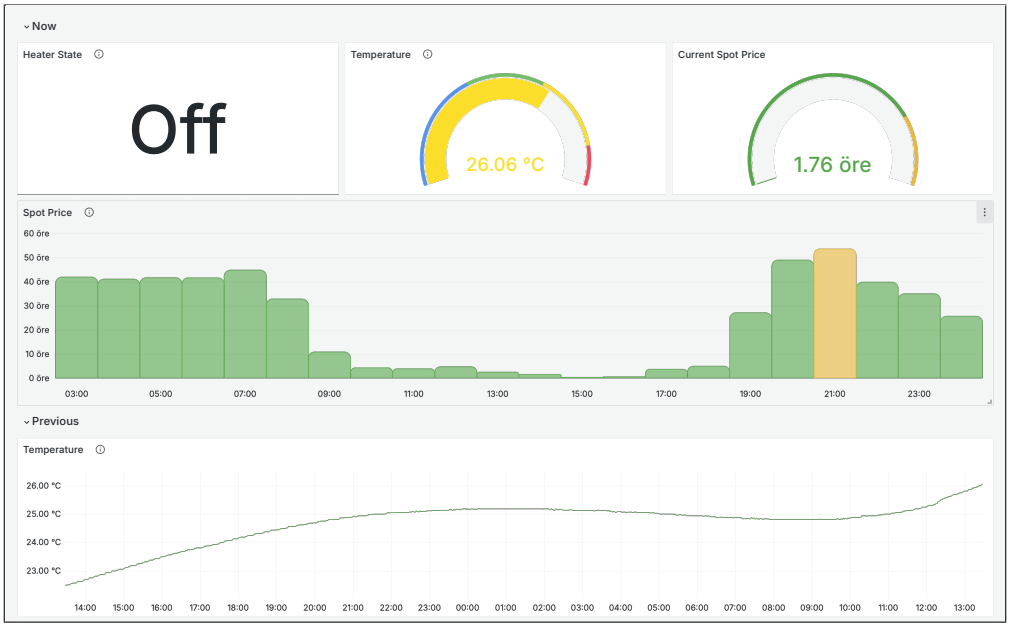
Figur 22: På panelens programflik kunde användaren stänga av och sätta på kontrollsystemet. Genom att klicka på startknappen kördes Python-programmet som beskrevs i Avsnitt 3.3.2. Programmets utdata visades i en textruta under knappen.



Figur 23: För säkerhets skull skyddades kontrollpanelen av ett enkelt användarsystem, där det krävdes att gruppmedlemmarna loggade in med användarnamn och lösenord. En bild på formuläret visas i denna figur.



Figur 24: Det inbyggda gränssnittet i InfluxDB. Det erbjuder ett visuellt sätt att konstruera en förfrågan för att hämta data från ett specifikt tidsintervall. Det går också att skapa använda fönstring och välja hur fönstrets värde ska beräknas. Slutligen presenteras förfrågans data, och en möjlighet att spara resultatet som en fil till sin dator.



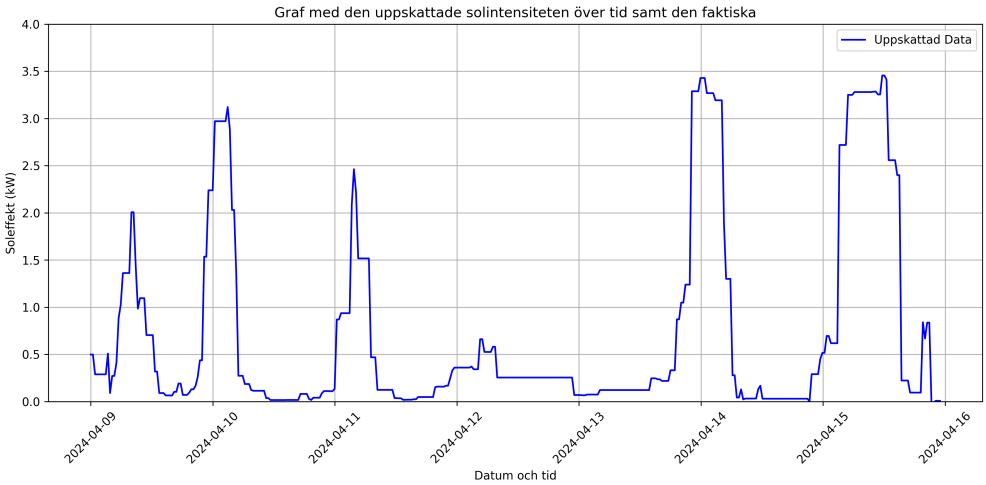
Figur 25: En skärmdump på hur data visualiserades med hjälp av *Grafana*. Värmefläktens status, inomhustemperaturen i byggnaden, elpriset för nuvarande timme, och så vidare. Sidan visade även historisk information från elmätarna och fläktens tillståndshistorik (visas ej i bilden). Detta gjorde det enkelt att snabbt få en överblick av många olika data.

4.3 Rebase

I detta underavsnitt presenteras prognosernas noggrannhet för de två olika prognosmodellerna som beskrevs i metod.

4.3.1 Prognos baserad på data från byggnaden

Prognosmodellen som utvecklades med platformen Rebase, för att göra uppskattningar av byggnadens framtida elproduktion, resulterade i den blåa grafen enligt Figur 26. Den data som modellen tränades på var uppmätt från byggnaden. Grafen beskriver den uppskattade solproduktionen över tid som användes i kontrollsystemet för att ta optimeringsbeslut.

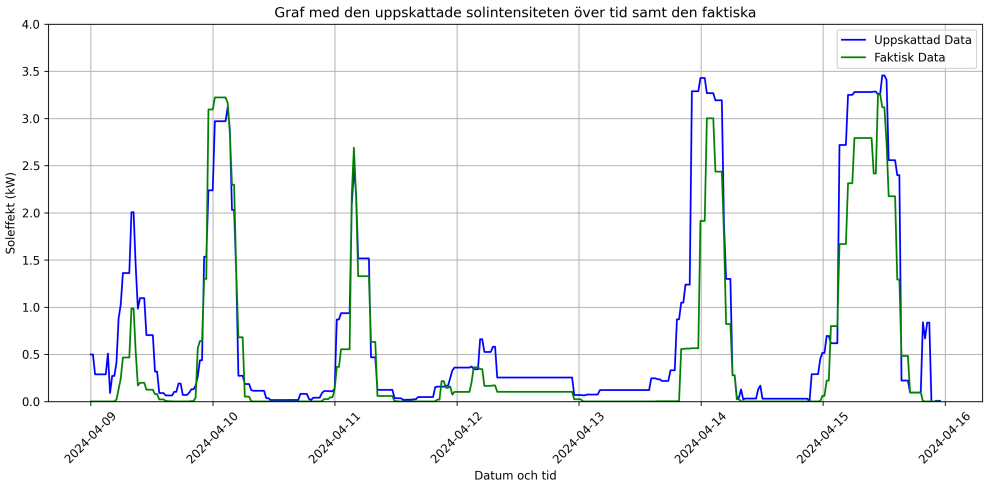


Figur 26: Graf över prognosen för solproduktionen över tid.

Prognosmodellen när den i efterhand jämförs med den verkliga datan illustreras i Figur 27. Denna figur visar skillnaden mellan den faktiska datan och den uppskattade, med ett *Mean absolute percentage error* på 54.7% samt ett *Normalised mean absolute error* på 7.68%. Dessa beräknades av Rebase enligt (2.6), samt med normaliseringsmetoden som förklaras i Avsnitt 2.2 [22].

Tabell 2: Beräknad noggrannhet i Rebase av prognoserna med hjälp av MAPE och NMAE.

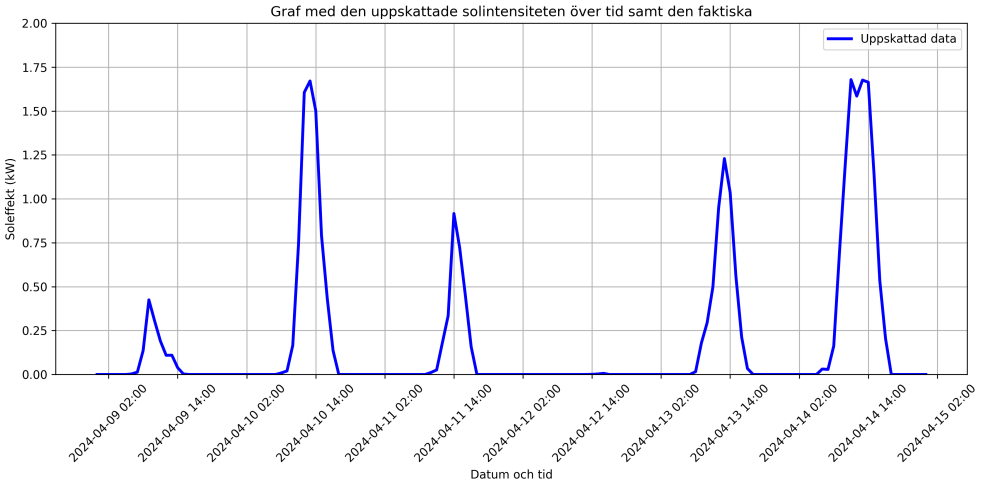
$NMAE$	$MAPE$
7.68%	54.7 %



Figur 27: Graf med prognosen över solintensiteten och den faktiska över tid.

4.3.2 Prognos baserad på data från Ellevio

Den Elleviobaserade produktionsprognosen ger också svagare prognosvärden än den baserad på insamlad data. Genom att studera Figur 33, som visar uppmätta värden, är det rimligt att tolka denna prognosmodell som något mindre bra, vilket också var uppfattningen i Avsnitt 3.1.3. I Figur 28 presenteras den prognosen för modellen tränad på data hämtad från Ellevio.



Figur 28: Graf med prognosen över solintensiteten.

4.4 Termisk modell

I Tabell 3 visas konstanterna som uppskattades i Avsnitt 3.1.2.

Tabell 3: Uppskattade parametrar för den termiska modellen.

R_{eq} (K/kW)	C_{air} (kWh/K)	$R_{in_{iso}}$(K/kW)	C_{inter} (kWh/K)
22.8	0.035	36	5

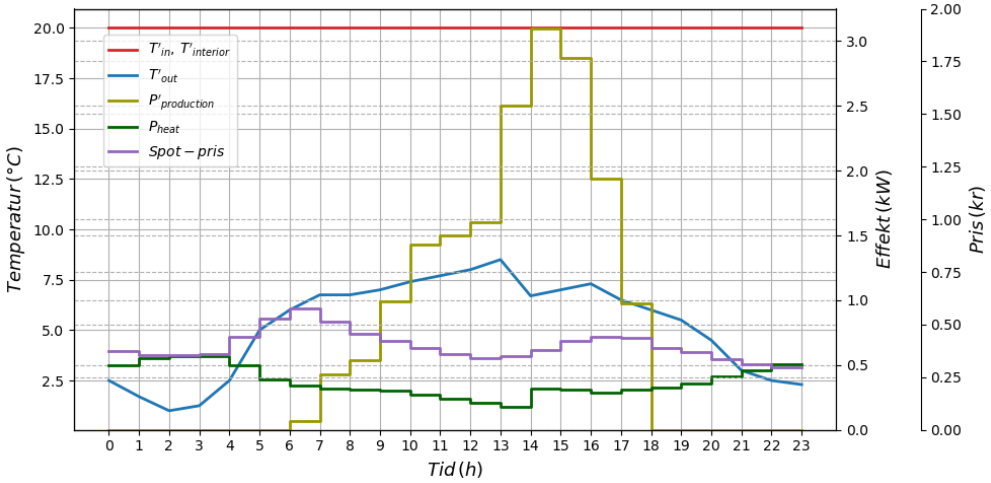
β	COP	$P_{utrustning}$ (kW)	P_{sol} (kW)
0.8	1	0.369	0

4.5 Tester

Denna underrubrik presenterar resultat från tester utförda med kontrollsystemet. Eftersom tester utfördes parallellt med utvecklingen så såg systemet inte likadant ut för samtliga tester. Skillnader redogörs för under respektive test. En rad felkällor har dessutom identifierats, exempelvis följde inte fläkten exakt det värmeschema som bestämts, speciellt för Avsnitt 4.5.2. Se Avsnitt 4.5.3 längre ner. Observera att symboler följt av ett $'$ hänvisar till en prognos eller av optimeringen beräknad variabel medans de utan innebär i realtid uppmätt data. $P'_{produktion}$ refererar alltså till Rebase-prognosen för solproduktion medans $P_{production}$ syftar till den uppmätta produktionen. P_{heat} skrivs utan $'$ då den datan faktiskt implementeras. Notera att kostnad i testernas kontext bara syftar på spotpriset, för import respektive export.

4.5.1 Test 2024-04-20

Under lördagen den 20:onde april utfördes ett test av den första iterationen av kontroll och optimerings-systemet. Optimeringen illustreras i Figur 29 med givna variabler och parametrar och går att studera i detalj i Tabell 4. Till detta test användes Rebase modellen baserad på insamlad data.



Figur 29: Optimerad effekt till fläkten varje timme, P_{heat}, för uppvärmning av byggnaden, tillsammans med prognoser för utomhustemperaturen, $T_{out}\prime$, produktionen från solpanelerna, $P_{production}\prime$ samt $Price$ som i figuren kallas *spotpris*. Också inomhus- och interiörtemperaturen, $T_{in}\prime$ och $T_{interior}\prime$ beräknad enligt den termiska modellen. Optimering utförd dagen innan.

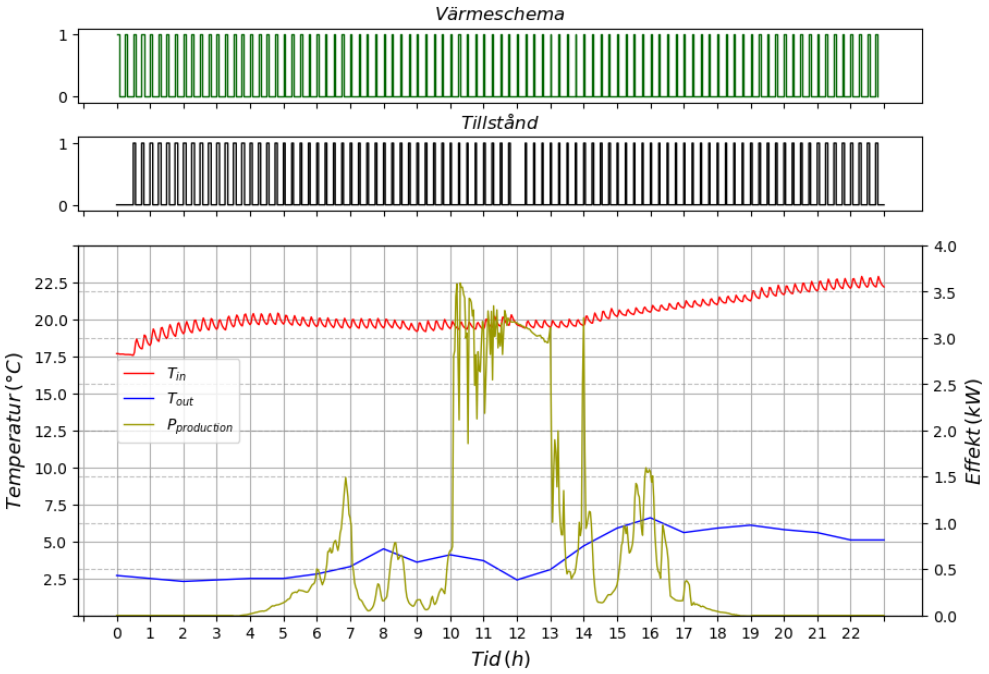
Tabell 4: Optimerade variabler T'_{in}, $T'_{interior}$, P_{heat} och $P'_{exchange}$. Resterande är parametervärden, där $Cost'$ är kostnaden alternativt vinsten för given timma. Temperaturer i $°C$, $P'_{exchange}$, $P'_{production}$ och P_{heat} i kWh, $Cost'$ i kr och $Price$ i kr/kWh.

Tid	T'_{in}	$T'_{interior}$	T'_{out}	$P'_{exchange}$	$P'_{production}$	P_{heat}	$Cost'$	$Price$
0	20.00	20.00	2.50	0.00	0.00	0.00	0.00	0.59
1	20.00	20.00	1.70	0.50	0.00	0.50	0.28	0.56
2	20.00	20.00	1.00	0.55	0.00	0.55	0.30	0.56
3	20.00	20.00	1.25	0.57	0.00	0.57	0.32	0.57
4	20.00	20.00	2.50	0.57	0.00	0.57	0.40	0.70
5	20.00	20.00	5.00	0.50	0.00	0.50	0.41	0.83
6	20.00	20.00	6.00	0.32	0.07	0.39	0.29	0.91
7	20.00	20.00	6.75	−0.09	0.43	0.34	−0.07	0.81
8	20.00	20.00	6.75	−0.22	0.54	0.32	−0.16	0.72
9	20.00	20.00	7.00	−0.68	0.99	0.31	−0.46	0.67
10	20.00	20.00	7.40	−1.13	1.43	0.30	−0.69	0.61
11	20.00	20.00	7.70	−1.23	1.50	0.27	−0.70	0.57
12	20.00	20.00	8.00	−1.36	1.60	0.24	−0.73	0.54
13	20.00	20.00	8.50	−2.30	2.50	0.21	−1.27	0.55
14	20.00	20.00	6.70	−2.92	3.10	0.18	−1.75	0.60
15	20.00	20.00	7.00	−2.56	2.87	0.32	−1.72	0.67
16	20.00	20.00	7.30	−1.63	1.94	0.31	−0.83	0.70
17	20.00	20.00	6.50	−0.69	0.98	0.29	−0.48	0.69
18	20.00	20.00	6.00	0.31	0.00	0.31	0.19	0.61
19	20.00	20.00	5.50	0.33	0.00	0.33	0.19	0.58
20	20.00	20.00	4.50	0.36	0.00	0.36	0.19	0.53
21	20.00	20.00	3.00	0.41	0.00	0.41	0.20	0.49
22	20.00	20.00	2.50	0.46	0.00	0.46	0.22	0.47
23	20.00	20.00	2.30	0.51	0.00	0.51	0.23	0.45

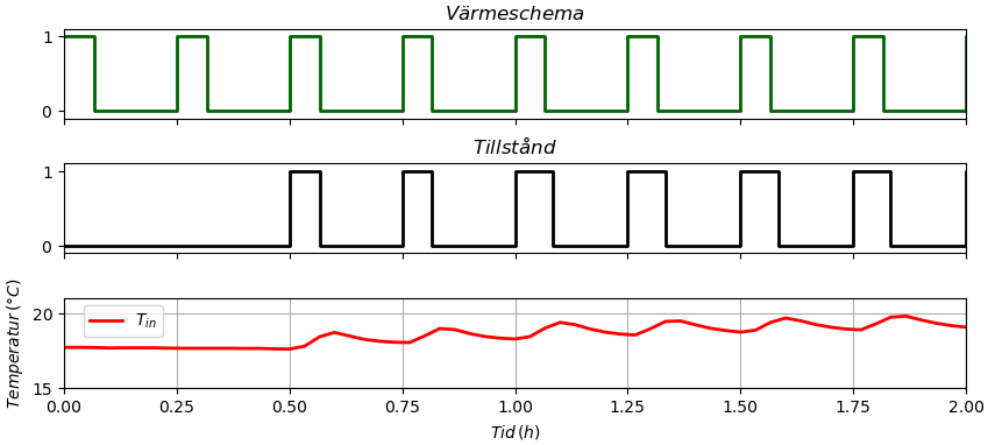
Som går att observera sträcker sig uppvärmingen inte över ett fullt dygn utan avslutas 23:00, vilket är ett resultat av feldesign. Den första timman för denna iteration av systemet initierade $T_{in'}$ och $T'_{interior}$ manuellt 00:00. Därför är startpunkten inte helt korrekt och i praktiken faller därför hela timmen bort för resterande variabler. Detta eftersom optimeringen vid stadiet var designat så T'_{in} vid en given timma, utom den första, berodde på P_{heat} för samma timma.

Optimeringen finner också genom den termiska modellen att $T'_{interior}$ är densamma som T'_{in} under dygnet, se Tabell 4, vilket är anledningen att de ges samma linje i Figur 29. Det ska också noteras att T'_{out} för detta experimentet är två timsteg fel då SMHI:s API använder UTC (Koordinerad universell tid) vilket upptäcktes i efterhand. $Price$ för testet var, liksom T'_{out}, inte anpassat från UTC och dessutom ifrån den 19:onde april eftersom mjukvaran för att plocka hem nästa dags elpris ej hade implementerats. $P'_{production}$ i optimeringen antas vara konstant varje timme, det vill säga att prognosvärdet för effekten en given timme behandlas som konstant för den timmen.

I Figur 30 visas uppmätta data för temperaturer och produktionen $P_{production}$ men också värmeschemat över det tillstånd fläkten faktiskt hade under testet. En fönsterbild över värmeschemat och tillståndet presenteras i Figur 31.



Figur 30: Värmeschema för fläkten: uppdelad för varje timma kvartsvis i av eller på stadier, 1 och 0, i enlighet med kontrollsystemet, samt det faktiska tillståndet hos fläkten under testet. Uppmätta temperaturer T_{in} och T_{out} från den installerade termometern och SMHI. $P_{production}$ är den faktiska solproduktionen, uppskattad från energimätare 1. Intervall mellan datapunkter är 1 minut förutom T_{out} som är timvis.



Figur 31: Fönsterbild av värmeschemat, tillståndet och inomhustemperaturen T_{in} för timmar 0-2.

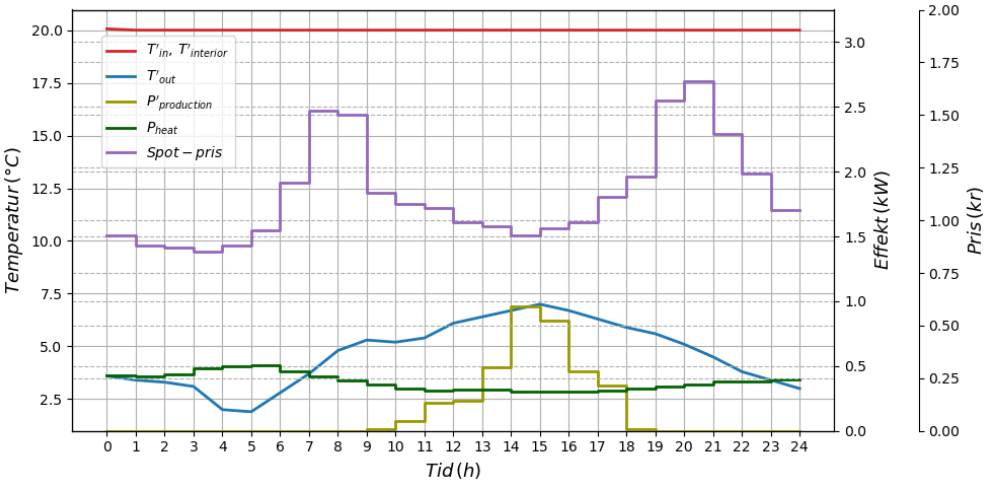
T_{in} var vid teststart ungefär 17.5 °C och verkar stabiliserat runt 20 °C för den första halvan av dagen för att senare stiga något. T_{out} är i Figur 30 för korrekt tidszon, UTC+2. Notera också att testet startade senare än planerat, se Figur 31. Utöver detta satte fläkten inte igång 00:00 och 12.00 som den skulle. Den totala kostnaden presenteras i Tabell 5.

Tabell 5: Summerade kostnader för testet. $Cost'_{tot}$ estimerad av optimeringen. $Cost_{tot}$ den faktiska kostnaden under testet, beräknad enligt (3.9). $Cost_{tot,even}$ kostnaden för en jämnt fördelad uppvärmning, beräknad med (3.10). Enhet kr.

$Cost'_{tot}$	$Cost_{tot}$	$Cost_{tot,even}$
−5.92	5.35	5.56

4.5.2 Test 2024-04-24

Onsdagen den 24:de april utfördes ett nytt test med de justeringar identifierade från resultatet i Avsnitt 4.5.1. Optimeringen förlängdes med en timma för att ta i beaktning initieringen av temperaturen och med detta köra hela dygnet. Vidare modifierades kontrollpanelen för att läsa av temperaturen vid körning av systemet för initiering. Den termiska modellen ändrades för att bero på förra timmens P_{heat}, enligt Avsnitt 2.7.1. Ytterligare förändringar inkluderade morgondagens $Price$ och en 2-timmars förflyttning av detta samt T'_{out} framåt för att hamna i korrekt tidszon, UTC+2. För detta test användes även den Elleviobaserade prognosen av $P'_{production}$, se Avsnitt 3.1.3 för detaljer. I Figur 32 och Tabell 6 visas som tidigare optimeringen.

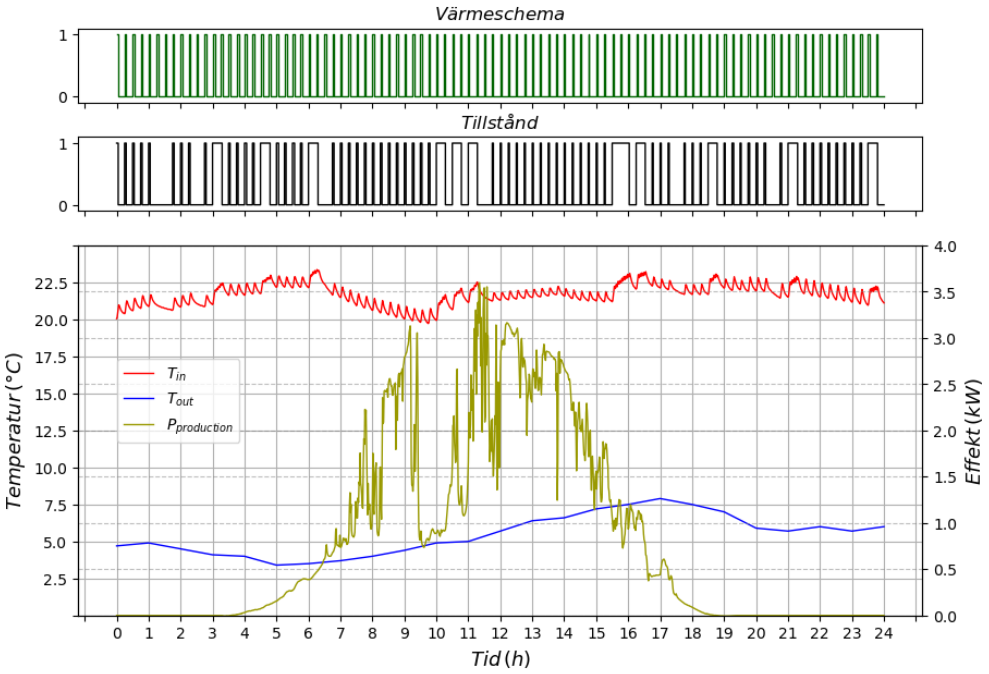


Figur 32: Optimerad effekt till fläkten varje timme, P_{heat}, för uppvärmning av byggnaden, tillsammans med prognoser för utomhustemperaturen, $T_{out}\prime$, produktionen från solpanelerna, $P_{production}\prime$ samt $Price$ som i figuren kallas $Spot - pris$. Också inomhus- och interiörtemperaturen, $T_{in}\prime$ och $T_{interior}\prime$ beräknad enligt den termiska modellen. Optimering utförd dagen innan.

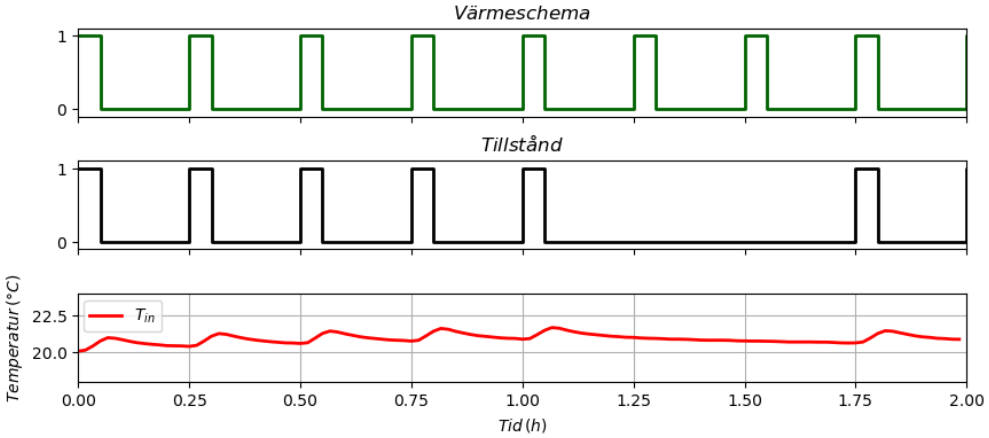
Tabell 6: Optimerade variabler T'_{in}, $T'_{interior}$, P_{heat} och $P'_{exchange}$. Resterande är parametervärden, där $Cost'$ är kostnaden alternativt vinsten för given timma. Temperaturer i $°C$, $P'_{exchange}$, $P'_{production}$ och P_{heat} i kWh, $Cost'$ i kr och $Price$ i kr/kWh.

Tid	T'_{in}	$T'_{interior}$	T'_{out}	$P'_{exchange}$	$P'_{production}$	P_{heat}	$Cost'$	$Price$
0	20.06	20.06	3.60	0.43	0.00	0.43	0.40	0.93
1	20.00	20.06	3.40	0.42	0.00	0.42	0.37	0.88
2	20.00	20.06	3.30	0.44	0.00	0.44	0.38	0.87
3	20.00	20.05	3.10	0.48	0.00	0.48	0.40	0.85
4	20.00	20.05	2.00	0.50	0.00	0.50	0.44	0.88
5	20.00	20.04	1.90	0.51	0.00	0.51	0.48	0.95
6	20.00	20.04	2.80	0.46	0.00	0.46	0.54	1.18
7	20.00	20.04	3.70	0.42	0.00	0.42	0.64	1.52
8	20.00	20.03	4.80	0.39	0.00	0.39	0.59	1.50
9	20.00	20.03	5.30	0.35	0.01	0.36	0.39	1.13
10	20.00	20.03	5.20	0.24	0.08	0.33	0.26	1.08
11	20.00	20.03	5.40	0.08	0.22	0.31	0.09	1.06
12	20.00	20.02	6.10	0.10	0.23	0.32	0.09	0.99
13	20.00	20.02	6.40	−0.18	0.49	0.32	−0.17	0.97
14	20.00	20.02	6.70	−0.65	0.96	0.30	−0.61	0.93
15	20.00	20.02	7.00	−0.55	0.85	0.30	−0.53	0.96
16	20.00	20.02	6.70	−0.16	0.46	0.30	−0.16	0.99
17	20.00	20.02	6.30	−0.04	0.35	0.31	−0.04	1.11
18	20.00	20.01	5.90	0.32	0.01	0.33	0.39	1.21
19	20.00	20.01	5.60	0.34	0.00	0.34	0.54	1.57
20	20.00	20.01	5.10	0.36	0.00	0.36	0.61	1.66
21	20.00	20.01	4.50	0.38	0.00	0.38	0.53	1.41
22	20.00	20.01	3.80	0.38	0.00	0.38	0.47	1.22
23	20.00	20.01	3.40	0.40	0.00	0.40	0.42	1.05
24	20.00	20.01	3.00	0.00	0.00	0.00	0.00	0.00

Som går att observera är optimeringen förlängd en timme. Timme 0 representerar temperaturen 00:00 och timme 24 temperaturen 24:00.



Figur 33: Värmeschema för fläkten: uppdelad för varje timme kvartsvis i av eller på stadier, 1 och 0, i enlighet med kontrollsystemet, samt det faktiska tillståndet hos fläkten under testet. Uppmätta temperaturer T_{in} och T_{out} från den installerade termometern och SMHI. $P_{production}$ är den faktiska solproduktionen, uppskattad från energimätare 1. Intervall mellan datapunkter är 1 minut förutom T_{out} som är timvis.



Figur 34: Fönsterbild av värmeschemat, tillståndet och inomhustemperaturen T_{in} för timmar 0-2.

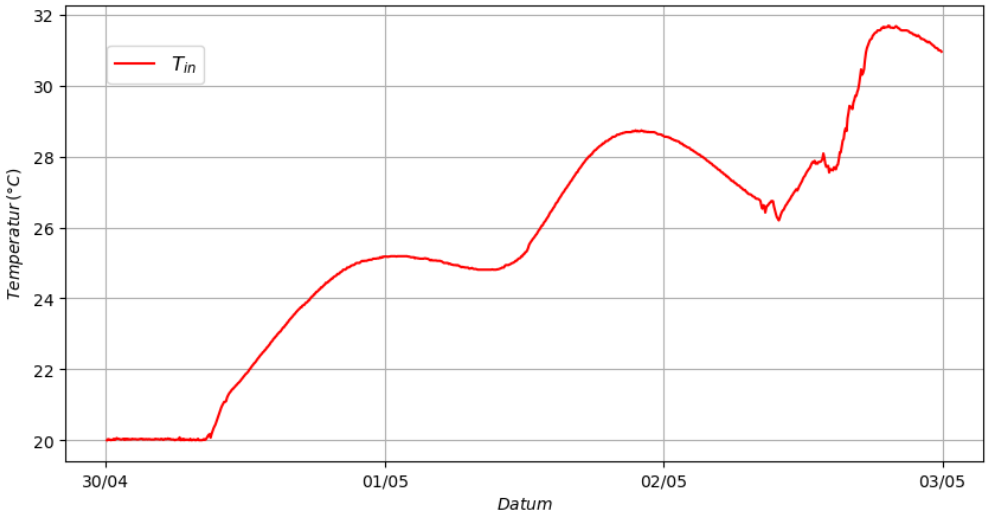
Till skillnad från det tidigare testet i Avsnitt 4.5.1 var kurvan för T_{in} ojämnare under detta test. Studerar man Figur 33 skiljer sig återigen tillståndet från värmeschemat, här starkare än tidigare.

Tabell 7: Summerade kostnader för testet. $Cost'_{tot}$ estimerad av optimeringen. $Cost_{tot}$ den faktiska kostnaden under testet, beräknad enligt (3.9). $Cost_{tot,even}$ kostnaden för en jämnt fördelad uppvärmning, beräknad med (3.10). Enhet kr.

$Cost'_{tot}$	$Cost_{tot}$	$Cost_{tot,even}$
6.52kr	10.15kr	10.19kr

4.5.3 Förhållanden under tester

För båda resultaten ska termostaten på anläggningens ventilationsfläkt varit inställd på 15 °C utan gruppens medvetande. I kontakt med Magnus Ellsén via e-mail fick gruppen på efterhand reda på att han ställt in detta 18/04 eftersom temperaturen i byggnaden ej tycktes gå lägre en 20 °C, även när värmefläkten var av. Gruppen resonerade vid samma tillfälle med Magnus Ellsén att temperaturen vid värmefläkten och termometern antagligen inte stämde överens, eftersom att termometern var fastsatt 2 m från byggnadens intag av kall utomhusluft. Ventilationens termostat ändrades till 25 °C 30/04 cirka 10:00 och efter det körde inte värmefläkten.



Figur 35: Temperaturutveckling i byggnaden efter ventilationen stängts av.

I Figur 35 visas den uppmätta temperaturen när termostaten på ventilationen ändrades och några dagar efter. Värt att nämna är att dessa datum såg högre temperaturer än de för testerna. På Vinga var utomhustemperaturen runt $20°C$ på dagen för dessa datum.

5 Diskussion

I detta avsnitt diskuteras resultaten, felkällor och rekommendationer för framtida arbeten.

5.1 Termisk modell

Den termiska modellen grundade sig i en 2R2C modell med de olika värmekällorna ihopklumpade som en källa där delar av värmen gick till luften och andra delar till innermaterial. Värdena i modellen anpassades efter Avsnitt 3.1.2, en anpassning som inte använde ett godhetsmått utan bara intuition. Även om modellen var ganska förenklad gick det att anpassa den till inomhustemperaturen. Under slutet av det tidsintervall som modellen anpassades efter sattes dock ventilationen ner till 15 °C vilket antagligen försämrade flera delar av den termiska modellen. Med tanke på hur varmt det blev efter att ventilationen stängdes av i Figur 35 anpassades antagligen den termiska resistansen i modellen som för låg. På liknande sätt blev modellens termiska kapacitans antagligen för låg då luftflödet från ventilationen gjorde dygnsvariationerna i temperaturen vid termometern större än i resten av byggnaden.

Från den stora toppen mellan den simulerade temperaturen och den uppmätta i Figur 11 kan det också märkas att något är fel. Toppen har sin källa i att effekten för byggnaden plötsligt ökade, varför effekten ökade just då och varför den inte påverkade temperaturen är okänt. En förklaring kan vara att förklaring kan vara att det var under samma tid som ventilationen sattes ner (vilket slog på ventilationen). Eftersom ventilationen agerar i vad som i enkla termer kan beskrivas som ett hål i väggen påverkar den termiska karaktären av byggnaden.

En bättre termisk modell hade antagligen kunnat fås av en mer detaljerad kunskap om hur byggnaden är uppbyggd, dess volym och isoleringsmaterial. Kunskap om dessa hade gett möjlighet att både beräkna mer korrekta värden av modellens parametrar och till en termisk modell av högra grad än 2R2C enligt Figur 8 vilket antagligen hade bidragit till en mer nogrann termisk modell och i sin tur även gett ett noggrannare kontrollsystem.

I den termiska modellen i Avsnitt 3.1.2 som anpassades på byggnaden togs inte byggnadens solinstrålning i beaktning. Att solinstrålningen inte används resulterar rimligtvis i en annorlunda inomhustemperatur än vad simuleringen uppskattar att det kommer vara. Detta illustreras i Figur 29 där det visas hur P_{heat} minskar under andra halvan av dagen då solen strålar som starkast. Figur 30 visar även en ökad solproduktion samt en högre utomhustemperatur under dessa timmar. Att solinstrålningen inte implementerades i den termiska modellen kan resultera i felberäkningar där den totala värmeffekten som bidrar till uppvärmningen blir för stor. För att erhålla en mer nogrann termisk modell borde effekten från solinstrålningen även implementerats i modellen då dess påverkan på inomhustemperaturen kan vara stor.

Det ska noteras att denna uppenbarelse antagligen tyder på en dåligt anpassad modell, utöver problemet med ventilationen, eftersom ett test med ventilationen avstängd hade sett temperaturen stiga, likt skeendet i Figur 35. Alltjämt uppskattades den termiska modellen, beskrivet i Avsnitt 3.1.2, med hjälp av anläggningsdata från perioder där ventilationen var inställd på 25 °C.

5.1.1 Byggnaden

En av de större felkällorna arbetet hade var mätutrustningens placering. Att termometern var placerad brevid intaget för ventilationen och i rak väg för den kalla luft från utsidan påverkade detta antagligen negativt både under anpassningen av den termiska modellen och körandet av

kontrollsystemet. Att termometern blev påverkad av ventilationen ledde antagligen till en mindre säker modell som uppskattade både den termiska resistansen samt värmekapacitetens fel.

Det kan även röra sig om problem i värmefläktens termostat. Av dess karaktär blir luften nära fläkten varmare än den övriga byggnaden och termostaten kan möjligtvis bli lurad till att stänga av fläkten utan att temperaturen är för hög.

I Figur 34 syns att fläktens schema skiljer sig från den på/av signal som skickas till den och fläkten förblir av när den borde vara på. Skillnaden av placering mellan fläkten och termometern förklarar antagligen skillnaden i fläktens tillstånd och värmeschemat, eftersom värmefläktens egna termostat var inställd för att reglera runt 30 °C, en högre temperatur än kontrollsystemet borde styrt den emot. Vi misstänker att fläktens låga placering i byggnaden gjorde att den var uppe i den på termostaten inställda temperaturen, samtidigt som termometern mätte lägre på grund av att den kalla luften från ventilationen. Figur 35 visar T_{in} uppmätt av termometern mellan 30/04 och 03/05. Enligt termometern ökade temperaturen från 20 °C till 30 °C, som är orimligt varmt för en inomhusbyggnad. Detta kan bero på att grafen i Figur 35 både innan och efter avstängningen representerar den verkliga temperaturen inomhus och att termometern mäter rätt. Vi tror detta är osannolikt från de observationer som gjorts i Figur 34.

En annan mer stokastisk källa till att byggnadens temperatur inte stämde överens med förväntade värden kan vara värmeenergi från vindkraftverket och solpanelerna. När vindkraftverket, som bara är igång slumpmässiga tider när någon är ute och kör det på plats, bromsas så övergår den kinetiska energin till värme genom ett effektmotstånd inuti byggnaden Avsnitt 2.3. Då vindkraftverket kan producera mycket mer än solenergin kan denna stå för en stor effekt som inte mäts någonstans. Kraftelektroniken till solpanelerna har värmeförluster och sitter också inne i byggnaden. Det är okänt hur mycket båda dessa bidrar vid olika tidpunkter till temperaturen.

5.2 Rebase-prognoser

Rebase prognoserna med den uppmätta datan är inte helt identiska med den faktiska datan då det finns en felmarginal som presenterades i Avsnitt 4.3.1. *Mean absolute percentage error* på 54.7% som beräknades med Rebase programmet är väldigt högt och tyder på dålig noggrannhet i prognosen. Däremot som beskrivet i Avsnitt 2.2 är MAPE metoden inte lika nogrann och lämplig för användning vid den här typen av energiprognoser. Här är det bättre att utgå på *Normal mean absolute error* värdet på 7.68% från Tabell 2, då den klarar av att hantera låga och negativa värden vilket modellen uppskattade att det skulle vara. Då NMAE inte bara tittar på hur mycket prognosen skiljer sig från det faktiska värdet utan normaliserar felen med hänsyn till variationen i de faktiska värdena, som förklarat i Avsnitt 2.2. Optimalt vill vi ha en så låg felmarginal på prognoserna som möjligt då det bidrar till ett mer noggrant optimeringssystem. Däremot blir detta svårt då oavsett tidsspann ändras NMAE väldigt lite vilket förmodligen beror på ett oundvikligt fel vid utvecklandet av modellen i Rebase. Eftersom det bara rörde sig om en veckas dat i tränandet av modellen kan detta ha en påverkan på dess precision.

Även i användandet av prognosen finns det felorsaker. Jämför man $P_{production}$ i Figur 29 och Figur 30, Avsnitt 4.5.1 framgår det att precisionen är suboptimal. En viss problematik existerar i att behandla produktionsprognoserna som fasta för varje timme. I Figur 29 säger prognosen att $P'_{production}$ klockan 14:00 kommer ligga på 3 kW men som går att urskilja i Figur 30 är detta bara fallet tillfälligt. För majoriteten av timmen var $P_{production}$ betydligt lägre. Att använda timvisa prognoser var antagligen dåligt. Som nämndes i Avsnitt 3.1.3 fanns det inställningar för bättre

upplösta prognoser. Alltså finns det tillgängliga lösningar, men dessa kräver att optimeringen också ändrar sitt intervall.

Modellen som skapades med konsumtionsdata från Ellevio kunde inte skapa en prognos för den historiska datan, vilket gick för modellen tränad på insamlad data. Av okänd anledning gick det inte att få upp.

5.3 Uppskatta kostnaden

Eftersom kontrollsystemet bara tar i beaktande spotpriset ger det inte en ärlig bild av det verkliga scenariot, beskrivet i Avsnitt 2.6. Eftersom faktorer som påverkar temperaturen och elpriset skiljer sig varje dag är det också svårt att göra jämförelser med konventionella reglersystem, speciellt när det rör sig om två dagars insamlad data.

Ett alternativ hade varit att köra systemet under en längre period för att någorlunda eliminera osäkra faktorer, så som elpris och väder. I så fall hade förbrukningen under en månad kunnat jämföras med tidigare år från Ellevio-kontot, men eftersom kontrollsystemet blev körbart så sent och de tidigare felkällorna identifierades hann projektgruppen inte med detta.

Att jämföra med Ellevio är inte heller utan risk. Eftersom datan på Ellevio berör mer än bara uppvärmningen, det vill säga utrustning och en hel byggnad till, hade uppskattningar varit nödvändiga. Eftersom kontrollsystemet bara tar i beaktning spotpriset hade det också varit svårt att jämföra, med regler för skatt av import och export 2.6.

5.4 Optimeringsmodell

I både Tabell 5 och 7 går det att se att optimeringen verkar fungera, ur avseendet att den i lösningen av problemen verkar minimera kostnader, jämför $Cost'_{tot}$ och $Cost_{tot,even}$. $Cost_{tot,even}$ och $Cost_{tot}$, den faktiska kostnaden, känns dock lite väl hög. Vi misstänker att byggnaden, vilket är rimligt från Avsnitt 4.5.3 där det konstaterades att det värmts på för mycket, har bättre termiska förmågor än vad som antagits i den termiska modellen. Som tidigare diskuterades bidrog andra faktorer som Psol också till uppvärmningen vilket inte tas i beaktning i optimeringen.

I båda testerna valde optimeringen att minimera uppvärmningen när $P_{production}$ var stor för att kunna sälja den producerade elen. Detta är ett resultat av målfunktionen, se (3.1), som försöker minimera $P_{exchange}$. Designen av optimeringssystemet gör alltså att uppvärmningen snarare styr mot $P_{production}$ än $Price$, eller i alla fall båda två och hindrar alltså i viss grad det syfte kontrollsystemet från början var tänkt för. Med målet att minimera $P_{exchange}$ är det inte orimligt att istället säga att modellen försöker exportera så mycket som möjligt, eftersom $P_{exchange}$ inte är bunden till positiva tal, se (3.2).

Detta skapar problem som är tydliga i skillnaden mellan $Cost'_{tot}$ och $Cost_{tot}$. Som diskuterades i Avsnitt 5.2 varierar $P_{production}$ mycket varje timme. När prognosen inte är bra, eller för dåligt upplöst, uppstår alltså en osäkerhet som kostar uppvärmningen väldigt dyrt. $Cost_{tot}$ och $Cost_{tot,even}$ är i Tabell 5 och 7 nästan identiska, vilket tyder på att testerna inte uppfyllt syftet, i synnerhet då ett normalreglerat system kan anses smartare än den jämna uppvärmningen $Cost_{tot,even}$ representerar.

Eftersom den Elleviobaserade prognosmodellen som användes i Avsnitt 4.5.2 gav svagare värden

var biaseringen emot solproduktionen inte lika stor som i Avsnitt 4.5.2, vilket går att se i skillnaden mellan $Cost'_{tot}$ och $Cost_{tot}$ i Tabell 7, som här skiljde med strax under $4kr$, jämfört med de avrundade $10kr$ i det tidigare testet. Det går att tro att optimeringen hade försökt styra mer mot $Price$ och att resultatet således skulle vara bättre än det tidigare. Speciellt eftersom $Price$ varierade mer under detta test. Dessvärre var $Cost_{tot}$ och $Cost_{tot,even}$ återigen väldigt lika och optimeringen verkade inte se någon vinst i att värma på vid låga kostnader. Detta kan bero på problematiken i optimeringens design runt $P_{production}$, vilket även när biasering är mindre påverkar styrningen mot $Price$, eller en kombination av faktorer i systemet. Exempelvis är den termiska modellen också utsatt för osäkerheter, som tidigare diskuterades.

Med de osäkerheter systemet redan besitter i kombination med det faktum att värmeschemat avrundas till minuter är de lösningar optimeringen finner väldigt lik en där man håller en konstant uppvärmning under dygnet. Med bättre prognoser hade förmodligen $Cost_{tot}$ hamnat närmare $Cost'_{tot}$, eftersom mycket av den producerade elen hade exporterats, men eftersom detta ej är fallet lider systemet som följd.

Att styra mot både $Price$ och $P_{production}$ behöver dock inte vara en dålig sak och inte nödvändigtvis motverka flexibilitet. Ett system som väljer att sälja sin egenproduktion när elpriset är dyrt hjälper att balansera elnätet vid tider av hög trafik, och kan fortfarande undvika de högsta topparna om ingen produktion finns tillgänglig. Detta är hypotetiskt, men med bättre solprognoser och en bättre anpassad termisk modell hade resultaten kunnat se annorlunda ut.

5.5 Rekommendationer

Under projektets gång noterades flera förbättringsmöjligheter, och de presenteras i detta avsnitt som rekommendationer för vidare studier.

Det som påverkat projektets utgång mest var förseningen av datainsamlingssystemet. Det är därför fördelaktigt om det finns data tillgängligt i början av projektet, eller om datainsamlingen implementeras tidigt i projektet så att indata kan ges till solprognosen så tidigt som möjligt. Utomhustemperaturen gjorde det även svårt att uppskatta skillnader när datainsamlingen hade implementerats. Eftersom temperaturen i april generellt sätt är varmare än januari, gav detta utslag även i resultatet. Den minskade temperaturskillnaden försvårar kostnadsoptimeringen och minskar möjlighet att se tydliga resultat av uppvärmning med värmefläkten, eftersom temperaturen sjunker långsammare.

Vidare är en rekommendation att konstruera optimeringsmodellen så att parametrarna återkopplas i realtid. Den kan då jämföra sin prognos mot det faktiska värdet direkt, och justera efter detta. I samma spår kan man också använda kortare tidsintervall i både optimeringsmodellen och för prognoserna. I projektet har timvis tidsintervall använts för optimeringen och Rebase, men en rekommendation är att använda kvartsvis eller minutvis istället för ett noggrannare resultat.

Den termiska modellen är inte komplicerad att anpassa, men den behöver välinsamlad data över en längre tid för precision. Viktigast är antagligen att se till att den temperatur som insamlas är representativ av hela byggnadens medeltemperatur. Om det är osäkert vilken energi som går till att värma upp byggnaden gör det både modellanpassningen och tolkning av resultat betydligt svårare. Det är även fördelaktigt att konstruera den termiska modellen för olika tidsintervall från början.

Om data hämtas från källor där tidsintervallen skiljer förenklar det projektet väsentligt om datan sparas i en tidsdatabas eller förbehandlas på ett sätt så all data ligger på gemensamma tidsintervall.

6 Slutsats

Syftet med denna rapport var att undersöka möjligheterna med att minska energikostnaderna genom att utveckla ett kontrollsystem för uppvärmning. Ett ytterligare syfte var att samla historisk och realtids mätdata från, och om byggnaden, för att uppskatta den passiva uppvärmning och energikonsumtionen av byggnaden, men också för att kunna utveckla prognosmodeller för solcellerna. Från rapporten kan följande slutsatser dras:

Insamlingen av historisk och realtidsdata var möjligt, men tar tid. Det går att bygga en enkel termisk modell för att upskatta värmebehovet av byggnaden, men för att den ska stämma behövs bättre mätutrusning än vad som användes.

Hårdvara som inte är uppkopplad till ett datainsamlingssystem bör prioriteras, annars kan den datan förloras. InfluxDB fungerade bra som databas för att lagra tidsserier från olika system och jämföra dem.

Kontrollsystemet i kontexten av optimeringen tyder på att det går att skapa flexibilitet men är i praktiken beroende av prognoserna och den termiska modellen. Att tillåta optimeringen att sälja egenproducerad el för att minska kostnader satte särkskild biasering mot egenproduktionen. Eftersom solprognosen i synnerhet led av den timvisa upplösningen orsakade denna biasering stora skillnader mellan optimering och resultat.

Linjär programmering resulterade i en optimeringsmodell som gick att använda för att styra elementet för att minska kostnaderna.

Referenser

[1] S. Petersson. ”Flexibilitet i elsystemet.” (dec. 2020), [Online]. Tillgänglig: https://ei.se/bransch/flexibilitet-i-elsystemet (hämtad 2024-05-08).

[2] G. C. Calafiore och L. El Ghaoui, *Optimization Models*. Cambridge University Press, 2014, ss. 1–20.

[3] R. Fletcher, *Practical methods of optimization*, 2. utg. John Wiley & Sons, 2000, ss. 140–150, ISBN: 978-0-471-49463-8.

[4] M. L. Bynum m. fl., *Pyomo-optimization modeling in python*. Springer, 2021, vol. 67, s. 32.

[5] A. Makhorin. ”GLPK (GNU Linear Programming Kit).” (juni 2012), [Online]. Tillgänglig: https://www.gnu.org/software/glpk/ (hämtad 2024-05-08).

[6] Y. Wexler. ”Understanding the Benefits of NMAE over MAPE for Estimating Load Forecast Accuracy.” (juni 2023), [Online]. Tillgänglig: https://www.amperon.co/blog/understanding-the-benefits-of-nmae-over-mape-for-estimating-load-forecast-accuracy (hämtad 2024-04-28).

[7] A. R. Lubis, S. Prayudani, Y. Fatmi, M. Lubis och Al-Khowarizmi, ”MAPE accuracy of CPO Forecasting by Applying Fuzzy Time Series,” i *2021 8th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2021, ss. 370–373. DOI: 10.23919/EECSI53397.2021.9624303.

[8] Ellevio. ”Så funkar elmarknaden.” (), [Online]. Tillgänglig: https://www.ellevio.se/om-ellevio/det-har-gor-vi/elen-framtiden/sa-funkar-elmarknaden/ (hämtad 2024-04-11).

[9] *A43/A44 User Manual*, rev. D, ABB, dec. 2020, kap. 9.

[10] T. Kugelstadt, *The RS-485 Design Guide*, rev. D, Texas Instruments, maj 2021.

[11] *MAX481/MAX483/MAX485/MAX487–MAX491/MAX1487, Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers*, rev. 10, Maxim Integrated, sept. 2014.

[12] *MQTT Version 5.0, OASIS Standard*, OASIS, mars 2019.

[13] S. kraftnät. ”Hur sätts elpriset och vad är ett priskryss?” (2023), [Online]. Tillgänglig: https://www.svk.se/om-kraftsystemet/om-systemansvaret/verktyg-for-systemdrift/forbrukningsfrankoppling/sa-har-funkar-frankoppling-om-elen-inte-racker-till/hur-satts-elpriset-och-vad-ar-ett-priskryss/ (hämtad 2024-04-11).

[14] Energimarkandsinspektionen. ”Elhandelsavtal.” (febr. 2024), [Online]. Tillgänglig: https://ei.se/konsument/el/elavtal/elhandelsavtal (hämtad 2024-04-15).

[15] V. Horn. ”Nord Pool: Så fungerar elhandeln.” (dec. 2019), [Online]. Tillgänglig: https://el.se/nord-pool (hämtad 2024-05-07).

[16] K. Lindholm. ”Elproduktion.” (mars 2023), [Online]. Tillgänglig: https://www.energiforetagen.se/energifakta/elsystemet/produktion/ (hämtad 2024-05-07).

[17] Energimarkandsinspektionen. ”Energiskatt.” (jan. 2024), [Online]. Tillgänglig: https://ei.se/konsument/el/elmarknaden/energiskatt (hämtad 2024-04-15).

[18] S. F. Fux, A. Ashouri, M. J. Benz och L. Guzzella, ”EKF based self-adaptive thermal model for a passive house,” *Energy and Buildings*, vol. 68, ss. 811–817, 2014.

[19] H. Park, M. Ruellan, A. Bouvet, E. Monmasson och R. Bennacer, ”Thermal parameter identification of simplified building model with electric appliance,” i *11th International Conference on Electrical Power Quality and Utilisation*, 2011, ss. 1–6. DOI: 10.1109/EPQU.2011.6128822.

[20] W. Wei, X. Yan, Y. Ni, Y. Luo Fengzhang Zeng och R. Xu, ”Power system flexibility scheduling model for wind power integration considering heating system,” i *2017 IEEE Power & Energy Society General Meeting*, 2017, ss. 1–5. DOI: 10.1109/PESGM.2017.8274612.

[21] D. Steen, "Modelling of Demand Response in Distribution Systems," diss., Chalmers University of technology, 2014.
[22] ReBase. "rebase.energy." (2017), [Online]. Tillgänglig: https://www.rebase.energy/#pricing (hämtad 2024-04-11).
[23] ENTSO-E. "ENTSO-E transparancy platform API." (), [Online]. Tillgänglig: https://transparency.entsoe.eu/content/static_content/Static%20content/web%20api/Guide.html (hämtad 2024-04-15).
[24] SMHI. "API för väderprognosdata." (juni 2016), [Online]. Tillgänglig: https://www.smhi.se/data/oppna-data/meteorologiska-data/api-for-vaderprognosdata-1.34233 (hämtad 2024-04-12).
[25] Ellevio. "Ellevio | Start." (), [Online]. Tillgänglig: https://www.ellevio.se/ (hämtad 2024-05-08).
[26] SMHI. "Villkor för användning." (nov. 2021), [Online]. Tillgänglig: https://www.smhi.se/data/oppna-data/information-om-oppna-data/villkor-for-anvandning-1.30622 (hämtad 2024-04-26).
[27] SMHI. "Prognosuppföljning." (), [Online]. Tillgänglig: https://www.smhi.se/data/meteorologi/prognosuppfoljning (hämtad 2024-04-08).
[28] S. Petersson. "Jämförelse av korta temperaturprognoser från SMHI och Meteorologisk institutt med fokus på post-processingmetodikens betydelse för prognoskvaliteten." (2019), [Online]. Tillgänglig: https://www.diva-portal.org/smash/get/diva2:1315925/FULLTEXT01.pdf (hämtad 2024-04-08).
[29] JrtPec. "entsoe-py." (mars 2024), [Online]. Tillgänglig: https://github.com/EnergieID/entsoe-py (hämtad 2024-05-02).
[30] *Raspberry Pi 4 Model B Datasheet*, rev. 1.1, Raspberry Pi (Trading) Ltd., mars 2024, kap. 9.
[31] *DS18B20, Programmable Resolution 1-Wire Digital Thermometer*, Maxim Integrated, juli 2019.
[32] "InfluxDB," *InfluxData Inc.* (2024), [Online]. Tillgänglig: https://www.influxdata.com/products/influxdb/ (hämtad 2024-04-26).
[33] S. Raimbault. "libmodbus, A featureful and portable Open Source Modbus library." (2022), [Online]. Tillgänglig: https://libmodbus.org/reference/ (hämtad 2024-04-26).
[34] "Shelly 2 API Reference," *Shelly Europe Ltd.* (2023), [Online]. Tillgänglig: https://shelly-api-docs.shelly.cloud/gen1/#shelly2 (hämtad 2024-05-01).
[35] "Grafana, The open observability platform," *Grafana Labs.* (2024), [Online]. Tillgänglig: https://grafana.com/ (hämtad 2024-05-08).
[36] L. Fosse, *Modbus*, https://github.com/f0sse/eenx16-modbus-rs485, 2024.
[37] L. Fosse, *Control Panel*, https://github.com/f0sse/eenx16-control-panel, 2024.

A control.py

```
import sys
sys.dont_write_bytecode=True
from zoneinfo import ZoneInfo
from elpriscurrent import getCurrentPrice
from PowerProduction_Forecast import result_production_forecast
from SMHIpy import get_next_day_t_values
import pyomo
import threading
from pyomo.environ import *
import matplotlib.pyplot as plt
import numpy as np
from SleepUtils import sleep_until
from HeaterControl import HeaterControl
from datetime import datetime, timedelta
#python biblioteken som importeras ovan behövs för att köra scriptet, se
↪  dokumentationen för dom för hur man får ner dom

# använder linjäroptimering och kör över ett dygn
def optimized_control():
    # nuvarande temperatur (skrivs till stdin)
    current_temperature = 20.0 if sys.stdin.closed else
    ↪  float(sys.stdin.readline().strip())

    # kontrollpanelen skriver nuvarande temperatur till stdin varje sekund
    def handleInput():
        for line in sys.stdin:
            if not threading.main_thread().is_alive():
                return

    # läs buffern (i bakgrunden) så att den inte blir för stor
    stdinthread = threading.Thread(target=handleInput)
    stdinthread.start()

    # Current TimeZone:
    current_tz = ZoneInfo("Europe/Stockholm")

    date_and_prices = getCurrentPrice(True) # True betyder att vi får kommande
    ↪  dags priser, hade vi sagt False hade vi fått dagens priser
    spot_prices = date_and_prices[1]  #sparar enbart spot priset för varje
    ↪  timme, inte datumet

    #tar bort data från prognoserna som inte är till användning för
    ↪  optimeringsmodellen
    keys_to_remove = ['additional', 'type', 'site_name', 'ref_time']
    for key in keys_to_remove:
        if key in result_production_forecast:
            del result_production_forecast[key]
```

```
#formatet för result_el_demand och result_production_forecast är som
↪  nedanstående, ifall de behöver kallas på
#result_el_demand = {'additional': {}, 'forecast': [1.6652443959133436,
↪  1.6652443959133436, ...], 'ref_time': '2024-03-26T05:00Z', 'site_name':
↪  'Demand', 'type': 'prioritized', 'valid_time': ['2024-03-26T00:00Z',
↪  ...]}

t0 = get_next_day_t_values() #hämtar temperaturvärdena enligt SMHIs prognos
↪  för nästa dag, sparas i t0
t0_hour = [temp + 273.15 for temp in t0] #Konverterar till Kelvin

# Modell mimimera kostnad timma med heat-constraint experiment
model = ConcreteModel()

# Set med timmar
model.hours = Set(initialize=range(25))

# Parametrar
model.T_out = Param(model.hours, initialize=t0_hour, default = t0_hour[-1])
model.price = Param(model.hours, initialize=spot_prices, default = 0)
model.Pprod = Param(model.hours,
↪  initialize=result_production_forecast['forecast'], default = 0)

# Termisk modell, kommenterade värden är de som utgicks från i början av
↪  projektet
R_eq = 1.9*12#20 # uppskattning från davids 3.3 K/kW
C_air = 0.035#0.08 #kWh/kelvin
R_in_iso = 3*12#5
C_inter = 5#2.5 #Värmekappacitet för väggarna
Beta = 0.8 #Hur mycket av värmen inomhus som överförs till isolering
COP = 1 #coffiecina of pefomacen
Putrustning = 0.369 #kW för datorerna??
Psol = 0*20/1000 #kW

# Initiera
T_in_initial = current_temperature + 273.15 #Kelvin
T_iso_initial = current_temperature + 273.15 #Kelvin

# Variabler
# Temperatur i byggnaden för varje timme
model.T_in = Var(model.hours, within=Reals, initialize=T_in_initial)
# Temperatur i isoleringen för varje timme
model.T_interior = Var(model.hours, within=Reals, initialize=T_iso_initial)
# Värme
model.Heat = Var(model.hours, within=NonNegativeReals)
#Importerad/exporterad energi
model.Pexchange = Var(model.hours, within=Reals)

# Objective rule
```

```
def objective_rule(model):
    return sum(model.price[hour] * model.Pexchange[hour] for hour in
    ↪  model.hours)
model.objective = Objective(rule=objective_rule, sense=minimize)

#Termisk modell
def thermal_model_rule(model, hour):
    if hour == 0:
        return model.T_in[hour] == T_in_initial
    return model.T_in[hour] == model.T_in[hour-1] - (model.T_in[hour-1] -
    ↪  model.T_out[hour-1]) / (R_eq * C_air) - (model.T_in[hour-1] -
    ↪  model.T_interior[hour-1]) / (R_in_iso * C_air) +
    ↪  (model.Heat[hour-1]*COP+Putrustning)*Beta/C_air
model.thermal_model = Constraint(model.hours, rule=thermal_model_rule)

def iso_model_rule(model, hour):
    if hour == 0:
        return model.T_interior[hour] == T_iso_initial
    return model.T_interior[hour] == model.T_interior[hour-1] -
    ↪  (model.T_interior[hour-1] - model.T_in[hour-1]) / (R_in_iso *
    ↪  C_inter) + (model.Heat[hour-1]*COP+Putrustning)*(1 - Beta)/C_inter
model.iso_model = Constraint(model.hours, rule=iso_model_rule)

#Temperature bounds
T_min = 20.0 + 273.15 #Kelvin
T_max = 25.0 + 273.15 #Kelvin
def temperature_bounds_rule(model, hour):
    return (T_min, model.T_in[hour], T_max)
model.temperature_bounds = Constraint(model.hours,
↪  rule=temperature_bounds_rule)

# Heat constraint
def heat_rule(model, hour):
    return model.Heat[hour] == model.Pexchange[hour] + model.Pprod[hour]
model.heat_constraint = Constraint(model.hours, rule=heat_rule)

# Pheat bounds
def pheat_bounds_rule(model, hour):
    return (0, model.Heat[hour], 2)
model.pheat_bounds = Constraint(model.hours, rule=pheat_bounds_rule)

#Använder GLPK (glpsol)
solver = SolverFactory('glpk')

#Kör glpsol --help för options
solver.options['tmlim'] = 60 # solution timeout [sekunder]
solver.options['memlim'] = 128 # solver memory limit [megabytes]

#Lös optimeringsproblemet (tee=True så att vi ser glpsol köra)
```

```
results = solver.solve(model, tee=True)

#Dubbelkolla att det finns en lösning
if results.solver.status == SolverStatus.ok and
↪  results.solver.termination_condition == TerminationCondition.optimal:
    #Print da shit
    print("{:<5} {:<10} {:<10} {:<10} {:<10} {:<10} {:<10}
    ↪  {:<10}".format('Timma', 'T_in (C)', 'T_interior (C)', 'Pexchange',
    ↪  'Pprod', 'Heat', 'Pris (kr)', 'Spot-pris (kr)'))

    total_cost = 0

    for hour in model.hours:
        cost = model.price[hour] * model.Pexchange[hour].value
        total_cost += cost
        T_in_C = model.T_in[hour].value - 273.15
        T_iso_C = model.T_interior[hour].value - 273.15
        price = model.price[hour]
        pimport = model.Pexchange[hour].value
        pprod = model.Pprod[hour]
        heat = model.Heat[hour].value
        print("{:<5} {:<10.2f} {:<10.2f} {:<10.2f} {:<10.2f} {:<10.2f}
        ↪  {:<10.2f} {:<10.2f}".format(hour, T_in_C, T_iso_C, pimport,
        ↪  pprod, heat, cost/100, price/100))

    print("Total cost: {:.2f} kr".format(total_cost / 100))
    #Sum Pexchange
    total_Heat = sum(model.Heat[hour].value for hour in model.hours)
    print("Total Heat: {:.2f} kW".format(total_Heat))
else:
    print("The solver did not find a feasible solution.")
    exit()

#Heater dictionary Det är denna ni vill anropa
heater_schedule = {}

fractions_minutes = []
fractions = [(model.Heat[hour].value / 2) * 60 for hour in model.hours]
for hour, fraction in enumerate(fractions):
    fractions_minutes.append(f'Hour {hour}: {fraction} minutes')

divided_minutes = []

for frac in fractions_minutes:
    parts = frac.split(':')[-1].split()
    total_minutes = float(parts[0])
    divided_minutes.append(total_minutes / 4)

# Morgondagens datum vid midnatt
```

```
    tomorrow = (datetime.now(current_tz) + timedelta(days=1)).replace(hour=0,
    ↪  minute=0, second=0, microsecond=0)

    current_state = None
    for hour, minutes in enumerate(divided_minutes): #hour=timme, 0-23, minutes
    ↪  = antal min varje timme, värdena!
        minutes = round(minutes)
        for quarter in range(4): #loopar 24 gånger, quarter = 0, 1, 2, 3
            start = hour * 60 + quarter * 15 #ex. 1*60 + 2*15 = 60 + 30
            end = start + minutes #ex 90 + 7
            for i in range(start, start + 15):  #start = 90, start + 15 = 105
            ↪  så från 0-105
                new_state = start <= i < end #om i ligger i intervallet 0-105
                ↪  så är new_state = true, annars false
                if new_state != current_state or current_state is None: #om
                ↪  new_state inte är current_state eller den ej har värde
                    timestamp = (tomorrow + timedelta(minutes=i))
                    heater_schedule[timestamp] = new_state
                    current_state = new_state

    with HeaterControl("172.31.68.4") as ctrl:
        ctrl.connect_blocking()
        wait_until_state = 10.0
        #skip the first hour
        dates = sorted(heater_schedule.keys())
        for date in dates:
            print(f"sleeping until: {date}")
            if not sleep_until(date):
                # returns false if date is in the past
                continue
            print(f"{date}: settings heater to {heater_schedule[date]}")
            ctrl.setstate(heater_schedule[date], wait_until_state)

    print(heater_schedule)
    #print(divided_minutes)

# kollar booleskt om den borde värma eller inte
def simple_control():
    # måltemperatur
    target = 20.0

    with HeaterControl("172.31.68.4") as ctrl:
        ctrl.connect_blocking()
        wait_until_state = 10.0

        naverage  = 10 # samples för moving average
        movingavg = [target for _ in range(naverage)]

        # nuvarande temperatur (skrivs till stdin varje sekund)
```

```
        for line in sys.stdin:
            try:
                temp = float(line.strip())
            except Exception:
                # hoppa över felformaterade värden
                continue
            else:
                # lägg till i moving average
                movingavg.insert(0, temp)
                movingavg.pop()

            newstate = (sum(movingavg) / naverage) < target

            oldstate = ctrl.state
            ctrl.state = newstate

            if newstate is not oldstate:
                oldstate = newstate
                date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                print(date, temp, newstate)

if __name__ == "__main__":
    simple_control()
    #optimized_control()
```

B SMHIpy.py

```
##
## Kod för prognosmodellen från SMHI
##
import requests
import base64
import json
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo

def get_next_day_t_values():
    entry = ("https://opendata-download-metfcst.smhi.se/api/category/pmp3g/"
    ↪  "version/2/geotype/point/lon/11.67984/lat/57.720835/data.json")
    txt = requests.get(entry)
    data = txt.json()

    zone = ZoneInfo("Europe/Stockholm")
    time_series = data.get("timeSeries", [])
    first_timestamp = datetime.strptime(time_series[0]['validTime'],
    ↪  '%Y-%m-%dT%H:%M:%SZ').astimezone(zone)
    next_day = first_timestamp.replace(hour=0, minute=0, second=0) +
    ↪  timedelta(days=1)
```

```
    day_after = next_day + timedelta(days=1)
    t0 = []

    for entry in time_series:
        if isinstance(entry, dict):
            current_timestamp = datetime.strptime(entry['validTime'],
            ↪  '%Y-%m-%dT%H:%M:%SZ').astimezone(zone)
            if next_day <= current_timestamp < day_after:
                for parameter in entry['parameters']:
                    if parameter['name'] == 't':
                        t0.append(parameter['values'][0])

    return t0
```

C elpriscurrent.py

```
##
## Kod för prognosmodellen för elpriset
##
import requests
import time
import xml.etree.ElementTree as ET #inte säkert mot felformaterade data
from datetime import timedelta, datetime, timezone
from zoneinfo import ZoneInfo

def getExchangeRate():
    url = "https://api.riksbank.se/swea/v1/Observations/Latest/sekeurpmi"
    r = requests.get(url, headers={"Accept": "application/json"})
    d = r.json()
    return d['value']

def getCurrentPrice(tomorrow):
    token = "<REDACTED>"
    endpoint = "https://web-api.tp.entsoe.eu/api?"
    domain = "10Y1001A1001A46L" #domän för SE3
    zone = ZoneInfo("Europe/Stockholm")
    starttime = datetime.now(zone).replace(hour=0, minute=0, second=0,
    ↪  microsecond=0)

    if tomorrow:
        starttime += timedelta(days=1)

    endtime = starttime + timedelta(days=1)

    # convert dates to accepted string format
    startstr = starttime.astimezone(timezone.utc).strftime("%Y%m%d%H%M")
    stopstr  = endtime.astimezone(timezone.utc).strftime("%Y%m%d%H%M")
```

```
    data =
    ↪  ("requests.get(f"{endpoint}securityToken={token}&documentType=A44&in_Domain="
    {domain}&out_Domain={domain}&periodStart={startstr}&periodEnd={stopstr}"
    )
    root = ET.fromstring(data.text)
    responseCheck = root.tag.split("_MarketDocument")[0]
    if "Acknowledgement" in responseCheck:
        raise Exception(f"Inga priser hittades för {'imorgon' if tomorrow else
        ↪  'idag'}")
    ns = root.tag.split("Publication_MarketDocument")[0]
    currency = (root.find(ns+'TimeSeries').find(ns+"currency_Unit.name").text)
    energytype =
    ↪  (root.find(ns+'TimeSeries').find(ns+"price_Measure_Unit.name").text)
    starttimeUTC =
    ↪  (root.find(ns+'TimeSeries').find(ns+"Period").find(ns+"timeInterval").find(ns+"start").te
    timelist = []

    # informative exchange rate from riksbanken
    rate = getExchangeRate()
    for i in (
    ↪  root.find(ns+'TimeSeries').find(ns+"Period").findall(ns+"Point")):
        # convert euro/MWh to öre/kWh
        timelist.append(float(i.find(ns+"price.amount").text) * rate / 10)
    return [starttimeUTC, timelist]
```

D HeaterControl.py

```
##
## Koden för styrning av värmefläkten
##
import os
import time
import ssl
import weakref
import paho.mqtt.client as mqtt

def decode_bytes(b):
        if type(b) == bytes:
                return str(b, encoding="ascii")
        if type(b) == str:
                return b
        return str(b)

CA = """-----BEGIN CERTIFICATE-----
<REDACTED>
-----END CERTIFICATE-----"""

def on_connect(client, heater, flags, reason, props):
```

```
        # Reason Codes:
        # - 0: Connection successful
        # - 1: Connection refused - incorrect protocol version
        # - 2: Connection refused - invalid client identifier
        # - 3: Connection refused - server unavailable
        # - 4: Connection refused - bad username or password
        # - 5: Connection refused - not authorised
        if reason == 0: # success
                heater._mqttc.subscribe(HeaterControl.TOPIC_STATE)
                heater._connected = True
        #if reason == 4 or reason == 5:
        #        raise Exception("MQTT authentication failure")

def on_connect_fail(client, heater):
        heater._connected = False

def on_disconnect(client, heater, flags, reason, props):
        heater._connected = False

def on_subscribe(client, heater, mid, reason_list, props):
        pass

def on_message(client, heater, message):
        topic   = decode_bytes(message.topic)
        payload = decode_bytes(message.payload)
        if topic == HeaterControl.TOPIC_STATE:
                if payload == "on":
                        heater._state = True
                if payload == "off":
                        heater._state = False

class HeaterControl(object):

        """
        HeaterControl: styrning av Shelly-pluggen över MQTT.
        """

        TOPIC_STATE   = "shellies/shellyplug-kandidat/relay/0"
        TOPIC_COMMAND = "shellies/shellyplug-kandidat/relay/0/command"

        def __init__(self, host, port = 8883, cid = "5672571", cleansession =
        ↪  False, *args, **kwargs):
                super().__init__(*args, **kwargs)
                username = os.environ.get("MQTT_USERNAME")
                password = os.environ.get("MQTT_PASSWORD")
                self._state = False
                self._connected = False
                self._looping = False
                self.host = host
```

```
                self.port = port
                self._mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2,
                ↪  cid, cleansession, protocol=mqtt.MQTTv311)
                self._mqttc.username_pw_set(username, password)
                self._mqttc.user_data_set(self)

                # set callbacks
                self._mqttc.on_connect      = on_connect
                self._mqttc.on_disconnect   = on_disconnect
                self._mqttc.on_connect_fail = on_connect_fail
                self._mqttc.on_message      = on_message
                self._mqttc.on_subscribe    = on_subscribe

                # TLS
                ctx = ssl.create_default_context(ssl.Purpose.SERVER_AUTH,
                ↪  cadata=CA)
                self._mqttc.tls_set_context(ctx)

                # Life-Cycle
                self._finalizer = weakref.finalize(self,
                ↪  HeaterControl._cleanup, self)

        def setstate(self, state, wait = 10.0, qos = 1):
                # it publishes state changes, so on_message
                # will update heater's state.
                #
                # qos:
                # 0: at most once
                # 1: at least once
                # 2: exactly once
                #
                if not self._looping or not self._connected:
                        return None
                if self._state == state:
                        return True
                info = self._mqttc.publish(HeaterControl.TOPIC_COMMAND, "on" if
                ↪  state else "off", qos)
                if wait > 0 and qos > 0:
                        info.wait_for_publish(wait)
                return info.is_published()

        @property
        def state(self):
                if not self._looping or not self._connected:
                        return None
                return self._state

        @state.setter
        def state(self, s):
```

```
                # it publishes state changes, so on_message
                # will update heater's state. set state
                # without ensuring delivery
                self.setstate(s, 0.0, 0)

        def connect(self):
                self._mqttc.connect_async(self.host, self.port, 21)
                self._mqttc.loop_start()
                self._looping = True

        def connect_blocking(self):
                if not self._looping:
                        self.connect()
                self._mqttc.reconnect()

        def disconnect(self):
                self._mqttc.loop_stop()
                self._mqttc.disconnect()
                self._looping = False

        def _cleanup(self):
                self._mqttc.loop_stop()
                self._mqttc.disconnect()

        def __exit__(self, exc_type, exc_value, traceback):
                self._finalizer()

        def __del__(self):
                self._finalizer()

        def __enter__(self):
                return self
```

E PowerProduction_Forecast.py

```
##
## Kod för prognosmodellen för power production
##
import requests
from datetime import datetime, timedelta

headers = {
  'GL-API-KEY': '<REDACTED>'
}

url =
↪  'https://api.rebase.energy/platform/v1/site/forecast/latest/d0fa895c-3375-4ce9-b407-a0c2da963
response = requests.get(url, headers=headers)
```

```
result = response.json()
#print(result)
from datetime import datetime, timedelta

# Calculate tomorrow's date
tomorrow = (datetime.now() + timedelta(days=1)).strftime('%Y-%m-%d')

tomorrow_valid_times = []
tomorrow_forecasts = []

for i, valid_time in enumerate(result['valid_time']):
  if valid_time.startswith(tomorrow):
    tomorrow_valid_times.append(valid_time)
    tomorrow_forecasts.append(result['forecast'][i])

result_production_forecast = {
  'additional': result['additional'],
  'forecast': tomorrow_forecasts,
  'ref_time': result['ref_time'],
  'site_name': result['site_name'],
  'type': result['type'],
  'valid_time': tomorrow_valid_times
}
```

F SleepUtils.py

```
##
## Kod för att sova till ett givet datum
##
import asyncio

from datetime import datetime
from time import sleep

def _secondsuntil(date: datetime):
        now = datetime.now(date.tzinfo)
        diff = date - now
        return diff.total_seconds()

async def sleep_until_async(date: datetime):
        delay = _secondsuntil(date)
        if delay > 0:
                await asyncio.sleep(delay)
                return True
        return False

def sleep_until(date: datetime):
```

```
        delay = _secondsuntil(date)
        if delay > 0:
                sleep(delay)
                return True
        return False
```

G temperature.py

```
import sys

sys.dont_write_bytecode = True

import asyncio
import os
import time
import glob

from math import floor
from os import EX_OSERR as EXIT_FAILURE
from datetime import datetime as dt

from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import ASYNCHRONOUS

async def every(s: int, func, *args, **kwargs):
        while True:
                now = dt.now()
                delay = s - now.microsecond / 1_000_000
                if delay:
                        await asyncio.sleep(delay)
                await func(*args, **kwargs)

async def ds18b20_read(f):
        raw_temp = int(f.readline().strip())
        while raw_temp == 85000:
                # power on / reset; try again
                f.seek(0)
                raw_temp = int(f.readline().strip())
        f.seek(0)
        return raw_temp / 1000.0

def read_to_point(reading, ts):
        t = dt.utcfromtimestamp(ts)
        return (
                Point("temperature")
                .tag("location", "björkö")
                .tag("sensor", "ds18b20")
                .field("temperature", reading)
```

```
                .time(t)
        )

async def reporter(client, writeapi, bucket, f):
        t = time.time()
        r = await ds18b20_read(f)
        p = read_to_point(r, t)
        writeapi.write(bucket=bucket, record=p)

if __name__ == "__main__":
        token  = os.environ.get("INFLUXDB_TOKEN")
        org    = "Kandidatarbete"
        url    = "https://8f.nu"
        bucket = "temperature"

        sensor = glob.glob("/sys/bus/w1/devices/28-*")

        if not len(sensor):
                print("No sensor available")
                sys.exit(EXIT_FAILURE)
        else:
                # assume only one sensor from now on
                sensor = sensor[0]

        try:
                with open(f"{sensor}/resolution", "w") as f:
                        #
                        # resolutions (lower gives faster reading):
                        #
                        #  9 -> 0.5
                        # 10 -> 0.25
                        # 11 -> 0.125
                        # 12 -> 0.0625
                        #
                        resolution = 12
                        f.write(f"{resolution}\n")

        except PermissionError:
                print("Could not change resolution to 0.0625 ...")

        with InfluxDBClient(url=url, token=token, org=org, debug=False) as
        ↪  client:
                with client.write_api(write_options=ASYNCHRONOUS) as writeapi:
                        with open(f"{sensor}/temperature") as f:
                                a = asyncio.get_event_loop()
                                a.create_task(every(1, reporter, client,
                                ↪  writeapi, bucket, f))
                                a.run_forever()
```

H price.py

```
import sys

sys.dont_write_bytecode = True

import pandas as pd
import asyncio
import os
import time

from math import floor
from os import EX_OSERR as EXIT_FAILURE
from datetime import datetime, timedelta
from zoneinfo import ZoneInfo

from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import ASYNCHRONOUS

from entsoe import EntsoePandasClient # pip install entsoe-py
from entsoe.mappings import Area

import requests


def eprint(*args, **kwargs):
        print(*args, file=sys.stderr, **kwargs)


def getExchangeRate():
        url = "https://api.riksbank.se/swea/v1/Observations/Latest/sekeurpmi"
        r = requests.get(url, headers={"Accept": "application/json"})
        d = r.json()
        return d['value']


async def every(s: int, func, *args, **kwargs):
        while True:
                now = datetime.now()
                delay = s - now.microsecond / 1_000_000
                if delay:
                        await asyncio.sleep(delay)
                await func(*args, **kwargs)


def get_price():
        client = EntsoePandasClient(api_key=os.environ.get("ENTSOE_TOKEN"))
        zone   = ZoneInfo("Europe/Stockholm")
        rate   = getExchangeRate()
```

```
        thishour = datetime.now(zone).replace(minute=0, second=0,
        ↪  microsecond=0)
        stopdate = thishour.replace(hour=0) + timedelta(days=1)

        if thishour.hour > 12:
                # also try to fetch day-ahead prices in the afternoon
                stopdate += timedelta(days=1)

        ts_start = pd.Timestamp(thishour.strftime("%Y%m%d"), tz=str(zone))
        ts_stop  = pd.Timestamp(stopdate.strftime("%Y%m%d"), tz=str(zone))

        s = client.query_day_ahead_prices(Area.SE_3, start=ts_start,
        ↪  end=ts_stop)

        # prices are reported in EUR / MWh. convert to öre / kWh
        return s * rate / 10


def to_point(t, price):
        return (
                Point("price")
                .tag("type", "entsoe")
                .field("hourprice", price)
                .time(t)
        )


async def reporter(client, writeapi, bucket):
        points = []

        try:
                today = get_price()
        except Exception as err:
                eprint(f"unable to fetch prices: {type(err)}")
                return

        for dt, price in zip(today.index.to_pydatetime(), today):
                points.append(to_point(dt, price))

        print(f"writing {len(points)} points")
        writeapi.write(bucket=bucket, record=points)


if __name__ == "__main__":
        token  = os.environ.get("INFLUXDB_TOKEN")
        org    = "Kandidatarbete"
        url    = "http://127.0.0.1:8086"
        bucket = "price"
```

```
interval = timedelta(hours=1).seconds

with InfluxDBClient(url=url, token=token, org=org, debug=False) as
↪  client:
        with client.write_api(write_options=ASYNCHRONOUS) as writeapi:
                a = asyncio.get_event_loop()
                a.create_task(every(interval, reporter, client,
                ↪  writeapi, bucket))
                a.run_forever()
```