STUDENTARBEID

HOVUDPROSJEKTRAPPORT

AUTOBRYGG

Deltakarar:

Mikal Jansen Berge, k.nr. 17
Kevin Alexander Østerhus, k.nr. 2
Guro Huus, k.nr.19
Torbjørg Huus, k.nr. 16

HO2-300 Bacheloroppgåve

VÅREN 2015

www.hisf.no

HØGSKULEN i SOGN OG FJORDANE

STUDENTRAPPORT

Boks 523 , 6803 FØRDE. Tlf: 57722500, Faks: 57722501 www.hisf.no

TITTEL	RAPPORTNR.	DATO
AutoBrygg	2.0	21.05.2015
PROSJEKTTITTEL	**TILGJENGE**	**TAL SIDER**
H02-300 Bacheloroppgåve	Opent	84

FORFATTARAR

Kevin Alexander Østerhus
Mikal Jansen Berge
Guro Huus
Torbjørg Huus

ANSVARLEGE RETTLEIARAR

Joar Sande

OPPDRAGSGJEVAR / SAMARBEIDSPARTNAR

AIN / Brygging AS Avd. Sagvåg

SAMANDRAG

I samarbeid med Brygging AS avd. Sagvåg har vi planlagt, skissert og tildels prosjektert eit automatisk ølbryggesystem. Vi har sett på behov av utstyr og jobba med samansetting av systemet slik at ein kan få best mogleg flyt i prosessen. Det har vorte laga prosessteikningar og utstyrsliste. Vi har laga programvarer for drift av systemet, med applikasjonar for bryggeprosessen og lagring av oppskrifter, og vi har oppretta ein database for lagring av oppskriftsfiler. Vi har òg laga ein enkel prototype av systemet.

SUMMARY

In cooperation with Brygging AS avd. Sagvåg we have planned, sketched and partly projected a automated brewing system. We have seen the needs of the equipment and worked with the composition of the system so that one can get the best possible flow in the process. It has been produced process engineering drawings and equipment list. We have created software for operation of the system, with applications for the brewing process and saving recipes, and we've created a database for storing recipe files. We have also created a simple prototype of the system.

EMNEORD

Programmering, HMI/GUI, temperatur- og nivåmåling, PID, styreskåp/tavle

Føreord

I faget HO2-300 Hovudprosjekt ved Høgskulen i Sogn og Fjordane skal studentane gjennomføre eit omfattande prosjektarbeid. Dette skal med anna omfatte planlegging, kostnadsanalyse, prosjektering, konstruksjon og utvikling av eventuell prototype[1]. Prosjektetarbeidet skal delast opp i forprosjekt og hovudprosjekt og denne rapporten gjeld hovudprosjekt.

Vi er fire studentar med ei stor interesse for ølbrygging og vi har jobba med ei oppgåve som omhandlar dette. Oppgåva har vi fått av Brygging AS avd. Sagvåg.

Vi vil retta ein stor takk til vår samarbeidspartnar i Brygging AS Avd. Sagvåg, Øystein Huus, for delt engasjement, hjelp og rettleiing undervegs i prosjektet.

Namn:	Dato, signatur:
Mikal Jansen Berge	_____
Kevin Alexander Østerhus	_____
Guro Huus	_____
Torbjørg Huus	_____

Innhaldsliste

Samandrag

Brygging AS avd. Sagvåg (heretter BS) vil bygge eit automatisert ølbryggesystem til bruk i verksemda. Formålet med dette systemet er i fyrste omgang å bruke det i forbinding med kurs, forum- og bryggesamlingar, men dei vil og sjå på høve for at kundar skal kunne bygge eit slikt system sjøl, der dei kan få råd i val av utstyr og høve til å kjøpe teikningar og programvare.

Bryggeprosessen består av fleire ledd. Grovt oppsummert er dette mesking, koking, kjøling, tapping på gjæringskar og gjæring, lagring (flaske/fat) og karbonering. Dei stega som skal inngå i det automatiserte systemet er prosessen frå mesking til tapping på gjæringskar.

I samarbeid med Brygging AS avd. Sagvåg har vi planlagt, skissert og tildels prosjektert eit automatisk ølbryggesystem. Vi har sett på behov av utstyr og jobba med samansetting av systemet slik at ein kan få best mogleg flyt i prosessen. Det har vorte laga prosessteikningar og utstyrsliste. Vi har laga programvarer for drift av systemet, med applikasjonar for bryggeprosessen og lagring av oppskrifter, og vi har oppretta ein database for lagring av oppskriftsfiler. Vi har òg laga ein enkel prototype av systemet.

Eit av delmåla i prosjektet var å bygge styringsskåp, men grunna forseinkingar fekk vi ikkje høve til å utføre dette. Då valde vi å heller lage prototypen.

Vi planla at den delen av programmet der oppskriftene skulle skrivast inn skulle vere ein veldig enkel funksjon. Av di det vart litt ventetid på bygging av skåp valde vi å nytta meir tid på denne delen av programmet og laga heller ein fullstendig applikasjon.

Tidsmessig har vi brukt ein del fleire timar enn planlagt. Vi kunne nok fint klart å avgrense oppgåva slik at vi heldt den planlagde timebruken, men vi har vert så engasjert og gira i prosjektarbeidet at vi rett og slett valde å ignorere dette litt. Det har ikkje gått ut over fokuset med å kome i mål med eit produkt i tide.

1.0 Forkortingar og definisjon av uttrykk

Mesking:
Dette er prosessen der knust malt blir trekt i vatn for å omdanna stivelse til sukkerstoff

Whirlpooling:
Ein prosess der ein sirkulerer vørteren slik at humle og trub samlar seg som ein klump i midten av tanken. Når vørteren så blir tappa ut vil mykje av dette grumset ligge igjen i tanken

MVC:
Model view controller - dette er ein måte å bygge opp program med grafiske element

B.K:
Boil Kettle - Kjele for koking av mesk

MLT:
Mash/lauter tun - Meskekjele

HLT:
Hot liquor tank - Tank for oppvarming av vatn

2.0 Innleiing

Innleiingsvis vil vi byrje med å nemne at prosjektgruppa er knytt til dette prosjektet privat og etter avslutta prosjektarbeid ved HiSF kjem vi til å fortsette å jobbe mot ei ferdigstilling av bryggesystemet.

Rapporten er omfattande skrive med mykje teknisk informasjon. Rapporten vart omfattande av fleire grunnar, men i hovudsak av di vi vil vise kva dette prosjektarbeidet har krevd av arbeid då vi har jobba veldig mykje, samstundes ynskjer vi at samarbeidspartnar skal få ei grundig innføring i korleis systemet er bygd opp og verkemåten. Dersom du ikkje har interesse av å lese den tekniske delen av rapporten kan du hoppe over kapittel 3.2 og 3.3 (side 18 til 69).

Prosjektgruppa består av studentane Mikal Jansen Berge, Kevin Alexander Østerhus, Guro Huus og Torbjørg Huus. Ansvarleg rettleiar er Joar Sande og AIN står som oppdragsgjevar. Samarbeidspartnaren vi utfører oppgåva for er Brygging AS avd. Sagvåg på Stord.

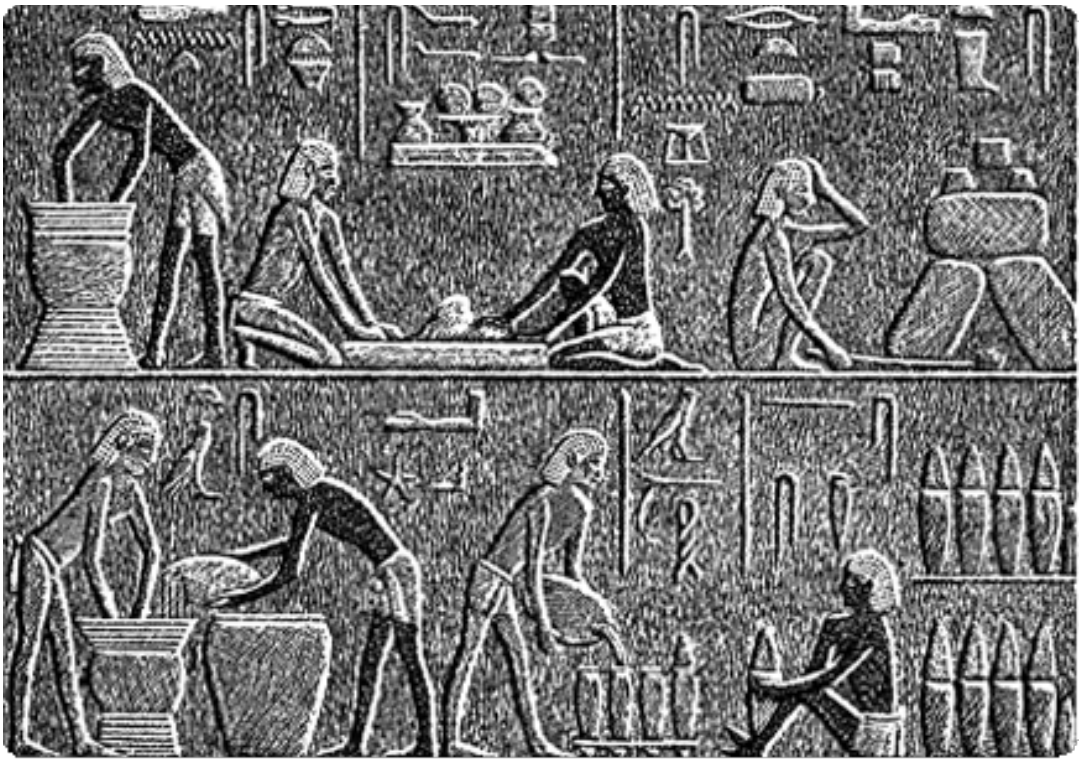
Brygging.no er ei verksemd som sel utstyr og ingrediensar til ølbrygging, primært via nettbutikk. Brygging AS avd. Sagvåg er ei avdeling i denne verksemda som driv med butikkutsal. Dei tilbyr òg kundane opplæring/kurs i bryggemetoden og driv forum.

2.1 Ølhistorie

Ingen veit med sikkerheit korleis ølet vart oppfunnet, men det har mest truleg skjedd ved (u)hell. Arkeologiske funn tyer på at øl vart laga i dei tidlege jordbrukssamfunna allereie for minst 7000 år sidan. Det fins ei 6000 år gamal steintavle i dagens Irak som visar mennesker som drikk noko som antas å vere øl frå ei bolle. Sidan den gong har øl vore ein vesentleg del av dei fleste befolkningsgruppers kultur og samfunn[2].

I middelalderen flytta ølets geografiske kjerneområde seg frå Midtausten til Europa, og i Nord- og Aust-Europa vart ølbrygging ein sentral del av alle familiar si hushaldning.
I Noreg har det òg vorte brygga øl så lenge det har vert drevet med jordbruk. Romaren Pytheas vart sannsynlegvis skjenka øl i Trøndelag i år 330 f.kr., men vi veit lite om starten på norsk ølbrygging, bortsett frå at vikingane drakk både øl og mjød. Frå ca år 1200 står det i Gulatingsloven at dersom ein bonde ikkje brygga øl til jul på tre år, skulle han fråtakast gård og gods[3].

Men desse lovane har endra seg mykje i tida etter og fram til i dag. Etter fyrste verdskrig var det periodar med forbodstid og det var ikkje før i 1999 at det var fritt fram for kven som helst å brygga øl sjølv i Noreg.



Figur 1 Gamal egyptisk steintavle som truleg viser ølbrygging

2.2 Bakgrunn

Heimebrygging av øl er i den siste tida blitt veldig populært og ein kan stadig sjå i aviser og på nett at dette er ein blomstrande hobby. Brygging er ikkje noko nytt, men den nye generasjonen bryggarar skil seg gjerne frå dei "tradisjonelle" bryggarane då ein kan sjå at fokuset har endra seg. Det blir lagt mykje vekt på å ha eit noko komplisert system, reinslegheit, ein nyttar seg av mange moderne hjelpemidlar ("gadgets") og øl som blir brygga skal vere presist i kontur og smak samanlikna med det ein vil oppnå. Ein kan sei at brygging er blitt veldig nerdete. Med den aukande interessa for brygging har markedet endra seg mykje. I dag har ein tilgang til uendeleg mange malt- og humletypar og ein kan brygge øl frå heile verda.

Gruppa ynskjer å utvikle ein automatisert ølbryggingsprosess for eit heimebryggeri. Med dette ynskjer vi å få til ein mest mogleg samanhengande prosess med minimal menneskeleg interferens.

Fordelar med ein meir automatisert bryggeprosess er fleire. Blant anna kan det gi mindre arbeid til bryggaren, ei meir nøyaktig temperaturregulering, vil gi repeterbarheit der ein kan enklare gjenskape «suksess-ølet» frå tidlegare brygging og så er det slikt at dess mindre menneskeleg berøring dess mindre forureining av ølet. Altså, eit enda betre sluttresultat!

Figur 2 Biletet viser råvarer og produkt

2.3 Problemstilling

Brygging AS avd. Sagvåg (heretter BS) vil bygge eit automatisert ølbryggesystem til bruk i verksemda. Formålet med dette systemet er i fyrste omgang å bruke det i forbinding med kurs, forum- og bryggesamlingar, men dei vil og sjå på høve for at kundar skal kunne bygge eit slikt system sjølv, der dei kan få råd i val av utstyr og høve til å kjøpe teikningar og programvare.

Bryggeprosessen består av fleire ledd. Grovt oppsummert er dette mesking, koking, kjøling, tapping på gjæringskar og gjæring, lagring (flaske/fat) og karbonering. Dei stega som skal inngå i det automatiserte systemet er prosessen frå mesking til tapping på gjæringskar.

Prosjektgruppa skal, i samarbeid med BS, planlegge og til dels prosjektere bryggesystemet. BS skal stå for den mekaniske delen og gruppa skal i hovudsak stå for instrumentering, elektronikk og programmering.

2.4 Hovudmål og delmål

Hovudmål
Prosjektgruppa skal planlegge og prosjektere eit automatisk ølbryggesystem. Vi skal i hovudsak jobbe med instrumentering, elektronikk og programmering. Vi vil óg sjå litt på kostnadane for å lage eit slikt system og korleis ein kan gjere det til eit kommersielt produkt. Oppgåva er lagt opp til å vere både teoretisk og praktisk.

Delmål
- Planlegge prosessen for systemet
    - Planlegging og dimensjonering for val av utstyr
    - Lage prosessteikning
    - Lage koplingsteikningar
- Bygge tavle/styringsskåp
- Programmere logikk
- Lage grensesnitt (UI)
- Lage database
- Testing
    - kommunikasjon mellom utstyr og program
    - kalibrering
- Kostnadsanalyse og undersøkingar

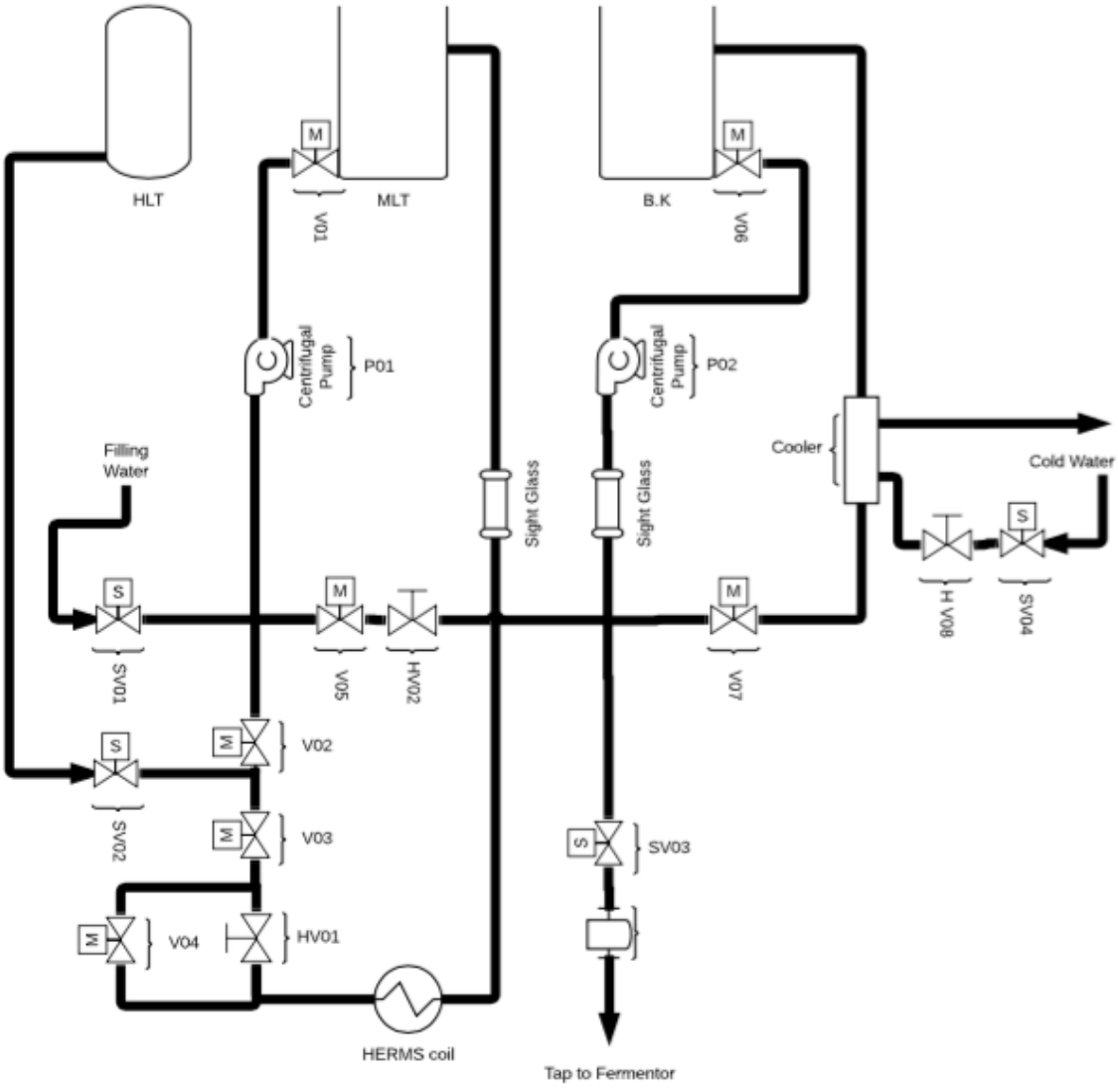
3.0 Hovuddel

3.1 Planlegging av prosessen

Ved utvikling av eit nytt bryggesystem vil ei viss erfaring i bryggekunsten vere ein viktig suksessfaktor. Vi hadde ved tidlegare prosjektarbeid allereie starta planlegginga av oppgåva, noko som gav oss ein god grobotn når vi skulle starte på hovudoppgåva. Vi lærte oss grunnleggande bryggeprinsipp og undersøkte ein del på kva utstyr vi kunne nytte til styring og elles i eit slikt systemet.

3.1.1 Prosessteikning

Prosessteikninga, vist i figur 3, har vi utvikla i samarbeid med samarbeidspartnar.



Figur 3 Prosessteikninga for systemet

Tryggleik har vert i fokus heile tida og vi har teke forholdsreglar. Vi har prosjektert med solenoid-ventilar på alle tilførslar og ved tappepunkt. Dersom noko uynskja skulle oppstå i prosessen vil vi kutte tilførsla av væske til og frå prosessen. Vi prosjekterte òg med ein HERMS-coil som samarbeidspartnar har erfaring med frå tidlegare system. Vi har nytta denne i vårt system for å indirekte varme opp mesket, dvs at ein ikkje har noko varmekilde/varmelement inne i meskebehaldaren (MLT). På den måten hindrar ein mogleg fastbrenning dersom noko skulle skje med elementet eller styringa, samt at ein får ein jamn temperatur i heile mesken. Det vart òg lagt ned mykje arbeid for å sikre pumpene i systemet. Pumpehusa må vera fylt med vatn for at impellaren i pumpa skal ha motstand når den byrjar å rotera. Dersom ikkje vil pumpa gå varm og kan då verta øydelagt. På pumpe 1 løyste vi dette ved at pumpa er avslått når mesketanken (MLT) vert fylt med vatn. Pumpe 2 er avslått når skyljeprosessen startar og pumpe 1 pumpar vatn gjennom pumpe 2 og opp i kokekjela (B.K). Det er viktig at ventilane som ligg føre pumpene ikkje er stengde når pumpene startar. Dette løyste vi ved hjelp av programmering i den automatiske delen og kopling i den manuelle delen.

Det vart lagt ned arbeid i funksjonaliteten til prosessen. Vi sette inn nokre handstyrte ventilar for å avgrense gjennomstrauminga i dei ulike stega i systemet. Når ein skal byrje å koke må ein tilsetje ein del vatn. Vatnet bør ha ein temperatur på rundt 70 - 80˚C. Når gruppa har brygga tidlegare har vi nytta to store gryter på komfyren for å varme opp vatnet. Det tar mykje tid og ein får som regel ikkje den stabile temperaturen ein ynskjer. Vi valde difor å integrere ein varmtvasstank til skyljeprosessen. Varmtvasstanken vil òg spare oss mykje tid ved andre høve, som ved vasking av systemet. Vi prosjekterte inn nokre sjåglas slik at brukar har høve til å sjå fargen på brygget. Kjøleprosessen er prosjektert inn som ein ekstern del.

3.1.2 Val av utstyr og komponentar i systemet

Når prosessteikningane var ferdige kunne vi starte å sjå på utstyr og komponentar for systemet. Vi har sett på utstyr og komponentar til skåpet og skåpdøra og komponentar i sjølve prosessen. Mekanisk utstyr som til dømes røyr og gryter har samarbeidspartnar ansvar for sjølv. Vi har laga ei uttømande komponentliste delt inn i skåp, skåpdør og koplingsmaterial samt for noko av utstyret i felt (Sjå vedlegg 9 for komponentliste).

3.1.3 Koplingsteikningar

Hovudstraum

Planlegging:
Vi planla at vi skulle forsyne varmelementa og pumpene frå ein 230V trefasa kontakt, som vidare skulle fordelast via sikringar. Kontrollsystemet skulle få ei eiga forsyning. Dersom sikringen går grunna for høg belastning vil kontrollsystemet framleis vere oppegåande, slik at når problemet er løyst vil bryggar kunne fortsette. Vi ville òg ha elektromagnetiske relé på utgangen til varmeelementa og pumpene slik at vi enkelt kan slå av desse utgangane med ein naudstopp.

Gjennomføring:
Forsyninga kjem inn frå ein 32A trefasa kontakt og blir kopla til ein trefasa sikring. Vidare vert fasane fordelt på tofasa sikringar, ein på 32A til varmeelementet på 5000W og ein på 16A til varmeelementet på 2500W. På utgangen av sikringane skal det monterast elektro-mekaniske relé. Det er sett opp ein eigen sikring til styresystemet som vert forsynt av ein tofasa stikkkontakt. Vidare vert spenninga omforma gjennom straumforsyninga og ein DC/DC-omformar.

DC-forsyning
Det vert nytta keramiske sikringar til å fordele DC-straumen. På denne måten vil ikkje ein feil i ein del av systemet påverke resten. Vi nytte òg desse som ein indikator på om det er noko gale med utstyret på ein spesifikk kurs.

Vi enda opp med ti 5V sikringar og ti 12V sikringar. Desse vart grovdimensjonert etter straumforbruket til dei ulike komponentane.

Brytarar

Planlegging:
Denne delen av teikningane krevde mykje planlegging av di vi ville få til ein overgang frå manuell styring til auto, noko som var krevjande å få til. Vi ville ha ein Manuell/Av/Auto vribrytar. I manuell modus ville vi slå av straumen til RPi og i auto ville vi slå av straumen til regulatorane samt andre manuelle brytarar. Oppdragsgjevar ville òg at det skulle vere mogleg å kople ut varmeelementa og pumpene, samt at pumpene skulle kunne køyre manuelt i auto. Det skal heller ikkje vere mogleg å køyre pumpene utan at ventilane på innsug er opne (som tidlegare beskrive i “planlegging av prosessen”). Vi ville ha brytarar som sette ventilstillinga for kvar del av prosessen, ein knapp til å opne ventilane og ein til å lukke. Vi vart òg einige om å nytte ein vribrytar til å slå av timeren.

Gjennomføring:
Manuell/Av/Auto-brytaren styrer relé som skrur på RPi og av regulatorane ved auto-stilling, og omvendt, og har ein posisjon der begge er av slått. Det er sett inn ein naudstoppbrytar som kutter straumen til alle “farlege” element. Til pumpene brukte vi tre-vegs vribrytarar for å møte krava om at det ikkje skal vere mogleg å aktivere pumpene utan at innsug-ventilane er opne. Ventilstillingsknappane set/reset nokre holde-relé og desse utgangane vert nytta vidare for å posisjonere dei aktuelle ventilane. Vi la til ein knapp som kan nyttast til å gi beskjed til RPi at den ikkje skal ta omsyn til ventilstillingane.

Ventil-relé
Sidan ventilane vert opna/lukka ved å snu på fasane måtte vi ha nokre relé til å gjere dette. Vi vil òg feilsikre dersom eit relé får signal om å opne og lukke samtidig ved at ventilen skal opne. Då kan bryggar overstyre ventilar om dette trengs.

Vi enda opp med å nytte fire relé for kvar motoriserte ventil, to for å opne og to for å lukke. Relea er laska saman på inngangane og viss alle fire relea blir slått inn vil ventilen opne. Relea får signal frå både den manuelle brytaren og RPi noko vi har tatt omsyn til i Man/Auto-brytaren. Til solenoid-ventilane i prosessen har vi òg nytta nokre relé til å forsyne desse med 12V.

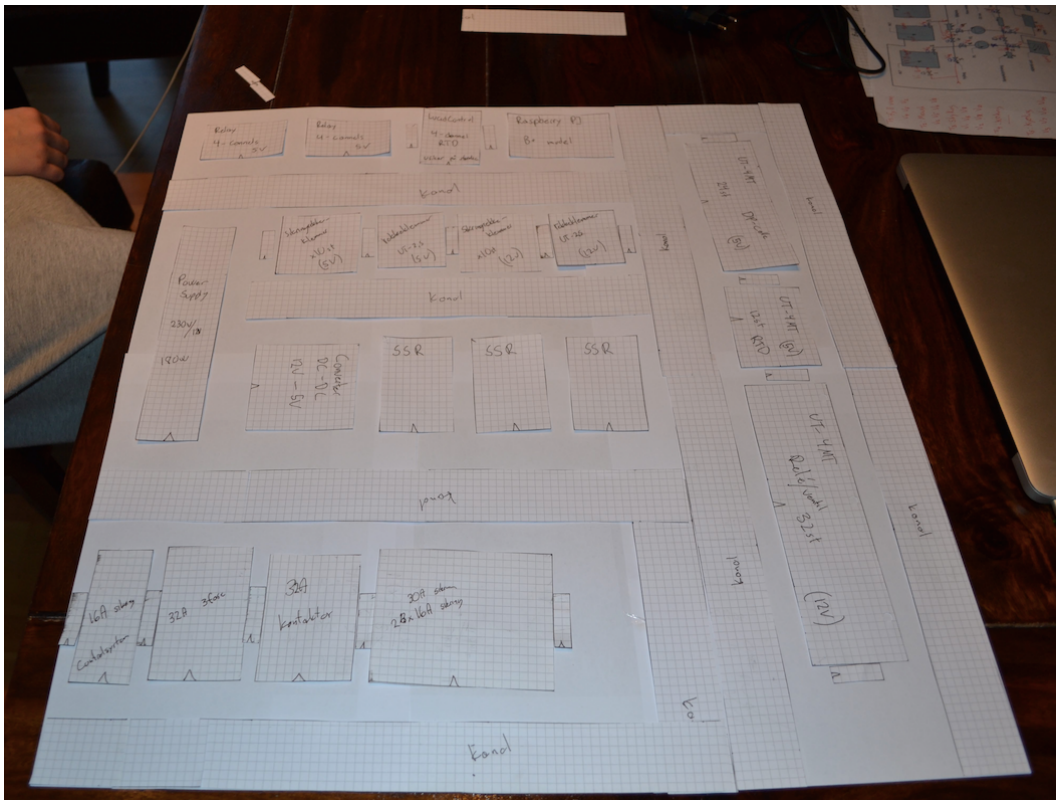
Misc
Vi måtte ha nokre relé til å setje pumper, skru på regulatorar og byte inngangskjelda til SSR’en. Vi har vald å nytta ventilstillingane til å illustrere ventilstillinga i flyt skjemaet ved hjelp av LED-lys. Det vart laga eigne teikningar til RPi-utgangane og -inngangane. Det er laga fullstendige termineringsteikningar for kopling av felt.

3.1.4 Styreskåp

Planlegging av skåpet
Det er ulike måtar ein kan prosjektere ei tavle på. Gruppa starta med å få eit oversyn over kva prosessen skulle innehalde av utstyr. Ut frå utstyret ute i feltet kan gruppa starte med å finne hovudkomponentane som skal monterast i tavla. Vi skal styre to varmeelement som vil sei at vi treng to SSR’ar. Skåpet skal forsynast med ei spenning på 230V som gjer at vi treng ei hovudsikring samt sikringar til ulike kursar. Ute i feltet opererer utstyret med ei spenning på 5V og 12V. Dette gjer at vi treng å spenne ned spenninga. Vi treng to omformarar, ein frå 230V til 12V og ein frå 12V til 5V. Gruppa hadde på førehand bestemt seg for å nytte Raspberry Pi(RPi) som ein kontroller. Denne modulen er eit bra val med tanke på pris og at det i tillegg finst mange tilleggskort[4].

Når hovudkomponentane var på plass starta gruppa på koplingsteikningane. Det er når du produserar koplingsteikningane du får eit skikkeleg oversyn over kor mange relé og rekkeklemmer som må prosjekterast inn. Vi får òg eit oversyn på kor mange inngangar og utgangar som trengs på Rasberry Pi-modulen. Dette vil gi oss ein fordel når vi skal byrje å programmere kontrollsystemet.

Når teikningane var på plass starta gruppa med å planlegge plasseringa i skåpet. Figur 4 viser korleis gruppa løyste dette. Vi fant måla på alle komponentane, teikna dei på ark og klippa dei ut. Vi sette saman fleire A4 ark som skulle representera monteringsplata til skåpet. Dette gjorde gruppa for å få eit oversyn over kor mykje plass vi hadde til rådigheit. Gruppa hadde fokus på å skilje dei ulike spenningane i skåpet. Vi hadde på førehand ein tanke om at vi ynskja å bruke eit skåp i storleiken 600x600mm. Under prosjektet har det vert ein del endringar. Figur 4 vart laga tidleg i prosjektet så dette er ikkje slik skåpet skal sjå ut, men framgangsmåten er den same.



Figur 4 Planlegging av plassering i skåp

Endringar

Under heile prosjektet har gruppa og samarbeidspartnar jobba tett. Dette har vi gjort gjennom møter og mail, spesielt i planleggingsprosessen. Det har vert viktig for begge partar slik at vi fekk ein god start på prosjektet.

Når ein startar på eit heilt nytt prosjekt vil det alltid vere rom for forbetringar. Det har medført at ferdigstillinga av teikningar og komponentliste har tatt meir tid enn planlagt. Dei største endringane vart gjort etter eit møte med samarbeidspartner den 14.03.15, møtereferatet kan du lese i vedlegg 6, protokoll 4.

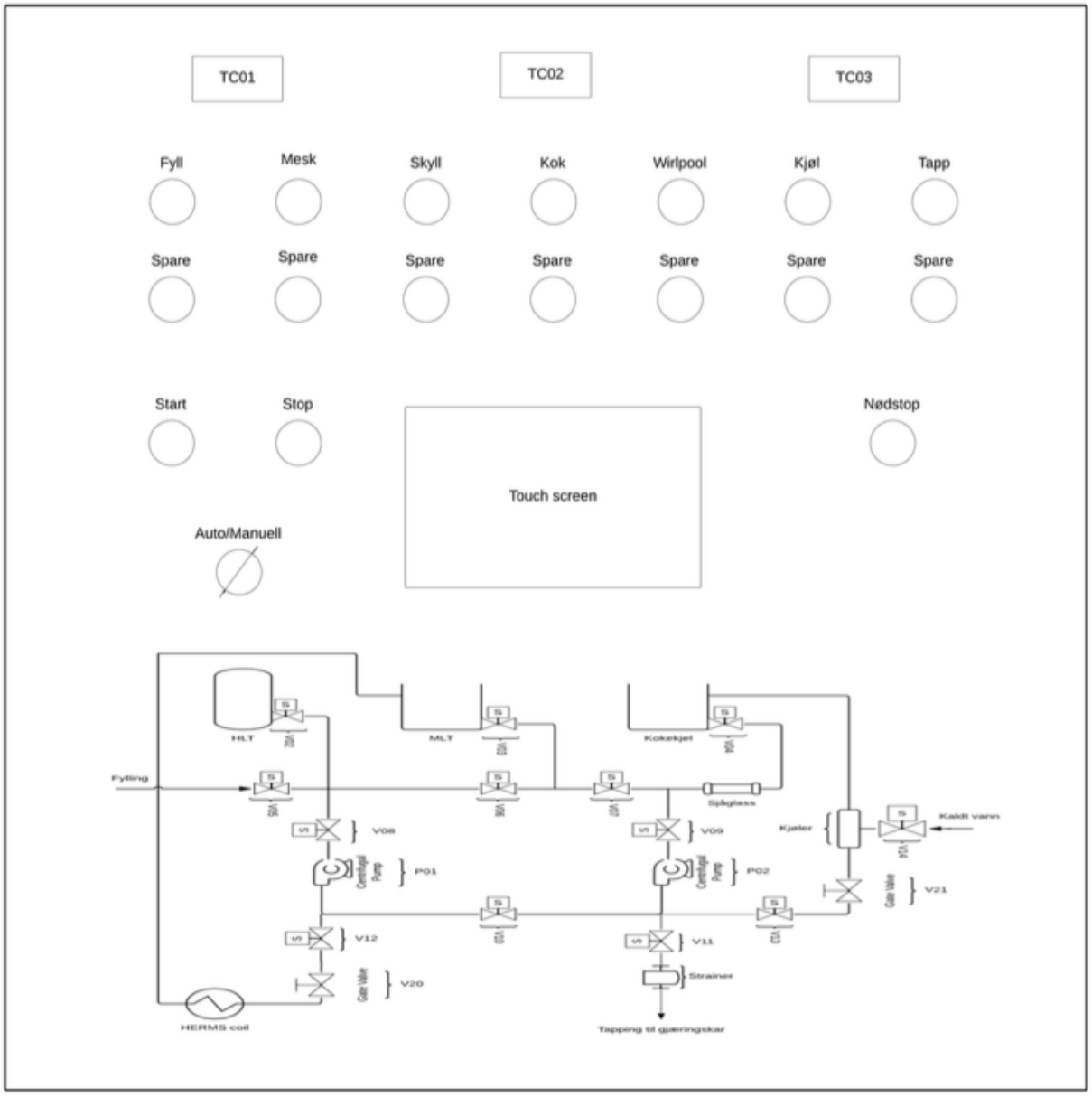
Under møte vart det bestemt av vi skulle byte ut ein del solinoid-ventilar til motoriserte ventilar med tilbakemelding. Dette gjorde at vi trengde fleire relé, rekkeklemmer og inngangar på RPI kortet. Det vart òg bestemt at vi skulle gjere varmtvasstanken til ein ekstern del. Dette medfører at vi ikkje treng styring av varmelementet til varmtvasstanken. Vi kunne då ta vekk ein automatsikring og ein SSR. Dei resterande to SSR'ane vart då flytta for å auke kjøleeffekten ved hjelp av ei vifte.

Figur 5 viser korleis den fyrste versjonen av skapdøra som gruppa prosjekterte såg ut. Vi hadde tre kontrollerer, ein til kvart varmeelement. Det vart sett opp ein brytar for kvart steg i prosessen. Vi hadde ein start- og stoppbrytar og ein touch-skjerm som skulle brukast til den automatiserte styringa.

Det vart store endringar på skapdøra av di samarbeidspartnar ynskja ein meir utfyllande funksjonalitet ved manuell styring. La oss starte frå toppen av figur 5. Som nemnt tidlegare vart varmvasstasken ein ekstern del. Dette bidrog til at vi kunne fjerne den eine kontrolleren. Den vart byta ut med ein tidtakar (timer) for å gi bryggaren høve til å sette ulike tider i kokeprosessen. Det vart sett opp to brytarar som skulle vere knytt til tidtakaren. Ein for å skru tidtakaren på og ein for å skru av alarmlyden.

Vidare hadde gruppa sett opp ein brytar for kvart steg i prosessen. Dersom bryggar trykka på eit av stega i prosessen ville då alt som var knytt til dette steget aktiverast, t.d ventilar, pumper og varmeelement. Dette ynskja samarbeidspartnaren å endre på ved å skilje styringa av dei ulike komponentane. Brytarane som representerte kvart steg i prosessen skulle bytast ut med to brytarar for å opne/lukke ventilane. Pumpene skulle få kvar sin trevegs vribrytar, Auto/Av/På. Varmeelementa skulle få ein tovegs vribrytar, Av/På. Brytarane start og stopp hadde etter endringa ikkje lenger noko funksjonalitet og vart difor fjerna.

Det siste som vart gjort var at vi sette inn ein «override» brytar. Den ignorerer ventilane si tilbakemelding om posisjon. Dette gir brukar høve til å overstyre systemet dersom ein ventil sluttar å verke.



Figur 5 Fyrste utkast av framsida til skåpet

3.2 Utstyr

Marknaden er proppfullt av ulikt utstyr og modular. Når gruppa skulle finne utstyr til systemet tok vi utgangspunkt i prosessteikningane. Ved prosjektering av styresystem må ein finne ut kva som er inkludert i prosessen. Prosessen inneheld pumper, ventilar, varmeelement og målarar. Vi må då prosjektere inn komponentar som til dømes SSR'ar, omformarar og styringsorgan.

Vi skal no sjå meir på utstyret vi har valt nytta i systemet.

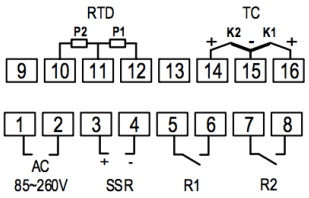
3.2.1 SMD-100 Kontroller

Figur 6 SMD-100

På systema våre har bryggaren to val. Han kan velje å køyre bryggeprosessen manuelt eller automatisk. I dette avsnittet skal vi ta for oss nokre av delane som inngår i prosessen dersom brukaren vel å styre brygginga manuelt.

Sjølv om bryggar vel å brygge i manuell modus vil noko av prosessen køyre sjølv (automatisk). Vi har integrert to PID-regulatorar til systemet slik at bryggar skal sleppe å skru av og på kokeplata og samstundes kontrollera temperaturen. Bryggar kan setje den ynskja temperaturen samt stille på ulike parameter. Dette vil bli utdjupa seinare i avsnittet. Når bryggar trykkar start vil regulatoren regulere inn den ynskja temperaturen. Regulatoren vil så halde på denne temperaturen til eit kriterium er oppfylt, noko brukar vel på førehand. I dag finst det mange PID-regulatorar på markedet med ulike funksjonar.

Kort om kontrolleren SMD-100



Figur 7 Terminaler

Kontrolleren er retta mot brukarar som skal overvaka og regulere temperatur i ein røykeomn[5]. Kontrolleren har moglegheit til å køyre opptil seks steg. Brukar må setje eit kriterium for kva tid kontrolleren skal gå over til neste steg. Denne kontrolleren har to ulike kriterium brukar kan velje mellom. Den fyrste og kanskje mest vanlege metoden er å setje ei tid. Eit døme på dette er at brukar ynskjer å halda temperaturen i omnen på 190 grader i ein time. Etter ein time vil kontrolleren gå over til neste steg, dersom dette er satt. Kontrolleren kan ha to Pt1000 element eller to av typen k termoelement, vist i figur 7. Ein må nytta to element av same type då det ikkje er mogleg å blande desse. Produsenten av denne kontrolleren har gitt brukar moglegheit til å bruke det andre elementet til å sjekke temperaturen inne i maten, som òg kan nyttas som eit kriterium for å gå til neste steg. Eit døme på dette er at brukar set ein temperatur på 190 grader i omnen og ein temperatur på det andre elementet, til dømes 150 grader. Når denne temperaturen er nådd vil kontrolleren gå til neste steg eller avslutte.

Figur 7 er henta frå databladet og viser alle tilkoplingsmoglegheitane. Det er mogleg å nytta to relé, R1 og R2. Desse reléa kan kun verta sett til dei ulike stega som ein har moglegheit for å setje opp. Eit døme på dette er at ein har eit tre-stegs program og på det tredje steget skal ei vifte byrje å kjøle. Dette vert stilla inn på kontrolleren og når det tredje steget byrjar vil relé R1 slå inn og vifta byrje å kjøle.

Kontroller må òg kunne styre eit varmeelement. Dette vert gjort gjennom ein SSR (Solid state relé). Kontrolleren nyttar ei spenning på 12V for å skru av og på SSR'en.

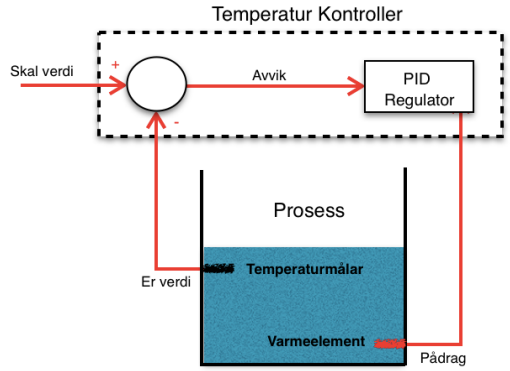
Kontrolleren kan lagre opp til åtte program.

Kvifor SMD-100

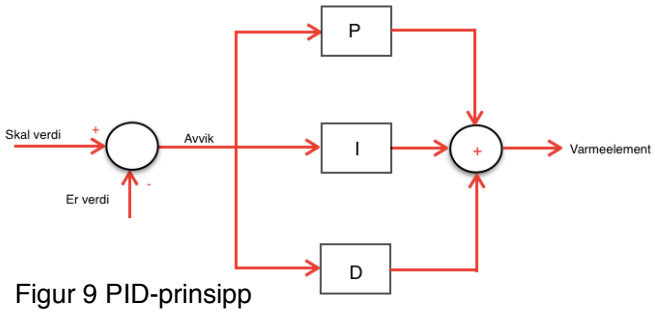
Denne kontrolleren har mange funksjonar og passar bra til brygging. Kontrolleren kan sette forskjellige steg, noko som passar utmerka til stegmesking. Vi kan nytta to temperaturmålarar på kontrolleren som gir høve for kaskaderegulering. Den har eit breitt måleområde frå -200 grader til 400 grader og kan lagre opp til åtte program. Rettleiing for korleis ein skal lage eit program finn ein i databladet. Dette er noko alle nye brukarar må gjennom og som regel trengs dette berre bli gjort ein gang.

PID-regulering

Som nemnt tidlegare styrer kontrolleren ein SSR for å regulere varmeelementet. Kontrolleren nyttar då ei PID-regulering, vist i figur 8. Prinsippet i seg sjølv er ganske enkel, men sjølve PID-behandlinga er noko vanskelegare. Reguleringa går kontinuerleg ved at kontrolleren henter prosess-verdien(er-verdien) og trekker denne frå skal-verdien, som er sett i kontrolleren. Verdien vi får er eit mål på kor stort avviket i prosessen er. Vidare vert verdien sendt til PID-regulatoren der den vert behandla. Utifrå resultatet vil kontrolleren justere på pådraget som i denne prosessen er eit varmeelement.



Figur 8 PID-regulering

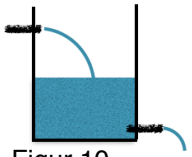


Figur 9 PID-prinsipp

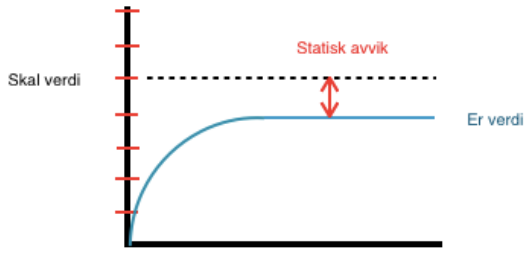
Ein PID-regulator består av dei tre matematiske metodane proporsjonal, integrering og derivering[6], vist i figur 9. Dei påverkar pådraget på ulike måtar. Det enklaste er å forklare ledda kvar for seg. Vi skal ikkje gå så nøye innpå matematikken som ligg bak kvart ledd, men korleis ledda påverka prosessen. På kontrollen kan ein sette tre ulike verdiar, Kp (forsterkning), Ti (integraltid) og Td (derivattid).

P-leddet

P-leddet tar avviket og ganger det med Kp. Brukar kan endre på Kp i kontrolleren. Dersom p-leddet får inn eit stort avvik vil kontrolleren gi mykje ut på pådraget og om det får inn eit lite avvik vil det gi mindre ut på pådraget. I dei fleste tilfella vil ikkje p-leddet åleine klare å oppnå skal-verdi. For å gi eit døme på dette kan vi ta utgangspunkt i figur 10. Vi har


Figur 10

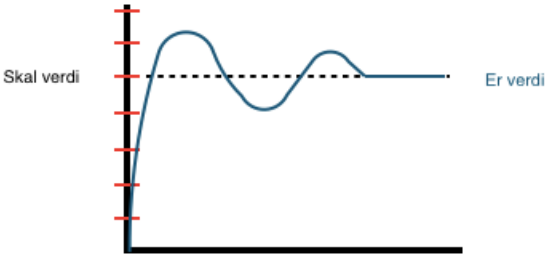
ein tank der det går like mykje væske ut av tanken som det kjem inn. Renn det 10l inn i tanken har ventilen fått ein verdi på 10 frå regulatoren. I tanken er det 40l og vi ynskjer at nivået skal vere 50l. Vi har då eit avvik på 10l. Avviket blir ganga med Kp som i dette tilfelle er satt til ein. Dette tyder at ventilen vil få den same verdien frå regulatoren og den vil ikkje endra på seg. Vi sit igjen med eit statisk avvik, vist i figur 11.



Figur 11 P-regulator

I-leddet

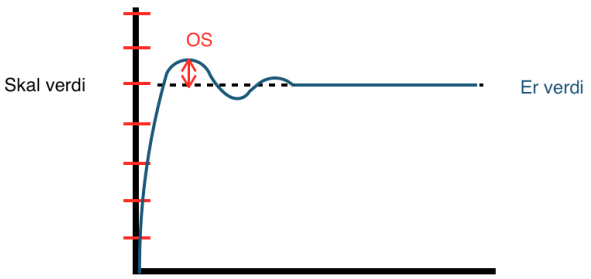
I-leddet i regulatoren har som oppgåve å fjerne det statiske avviket. Dette gjer den ved at den samlar opp avvik over ein viss tidsperiode, Ti. Denne tida har ein moglegheit for å endre i kontrolleren. Set ein denne verdien veldig høg i kontrolleren har ein i utgangspunktet kopla bort I-leddet. Figur 12 viser korleis prosessen vil fungere i praksis når Ti er stilt inn på ein optimal verdi.



Figur 12 PI-regulator

D-leddet

Dette leddet i regulatoren er det mest kompliserte og kan ofte bli valt bort. Då set ein Td lik null. Derimot om ein klarar å finne ein passande verdi vil d-leddet påverka oversvinget (OS) og kan med det redusera innsvingingstida, vist i figur 13. I motsetnad til I-leddet ser d-leddet fram i tid. Den ser på kor store endringar det er på avviket.



Figur 13 PID-regulator

Auto-tune
Når ein har kome så langt at ein skal byrje å kalibrera kontrolleren til anlegget kan dette bli gjort på ulike måtar. Som brukar har ein moglegheit til å setje opp PID-regulatoren manuelt. Det finst fleire rettleiingar på nettet om korleis dette skal gjerast, men dei fleste regulatorar idag har ein funksjon som heiter auto-tune. Då vil kontrolleren køyre systemet og setje PID-parametera som passar til denne prosessen.

3.2.2 Timer

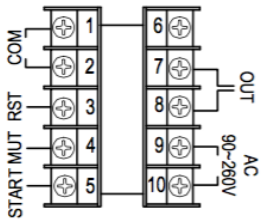
Kontrolleren vert integrert i systemet for at vi skal kunne regulere og sette ulike temperaturar. I ein bryggeprosess treng vi òg noko som kan halde kontroll på kva tid ein skal tilsetje ulike ingrediensar. Gruppa velde difor å integrera ein tidtakar(Timer) til prosjektet. Tidtakaren som gruppa har prosjektert med er av typen JSL-73B, vist i figur 14. Denne modulen er liten og kompakt og er spesielt tilpassa for bryggeprosessar.

Figur 14 Timer

Kort om JSL-73B
Denne modulen er veldig enkel å sette opp. Det er mogleg å sette opp til ni steg. Alt ein treng å vita er at det er tre ulike parametera som må settast[7]. I databladet vert desse kalla nE, Ad og TX.

nE - Set opp kor mange steg ein ynskjer å setje opp.
Ad - Her set ein varigheten på alarmlyden.
TX - Her set ein varigheten til kvart steg.

Figur 15 er henta frå databladet og viser modulen sine tilkoplingsmoglegheitar. På høgre side legg ein opp straumtilførselen og høgtalaren. Klemmene 7 og 8 er eit relé som vert slått inn når kvart steg er fullført. På venstre side kan ein legge opp tre knapper med ulike funksjonar. Desse knappane finn du òg på modulen sitt display, men dersom brukaren ynskjer å ha desse meir tilgjengeleg så er dette gjort med tilkoplingsklemmene 1 til 5.



Figur 15 Terminaler

Det er òg to display på modulen som viser ulike tider. Det nedre displayet viser kor lenge det er igjen av den totale koketida. Det andre displayet viser kor lenge det er igjen av det steget som modulen holder på å utføre.

3.2.3 Ventilar

Figur 16 Ventil

I prosessteikningane (vedlegg 12) har gruppa i samarbeid med samarbeidspartner prosjektert ein prosess med totalt fjorten ventilar. Plasseringa og ventiltypane er nøye gjennomtenkt for å sikre systemet og for å optimalisere bryggeprosessen. På alle innløp og utløp har vi vald å nytte solenoid-ventilar. Solenoid-ventilane som skal nyttast i prosjektet må tilførast spenning for at dei skal kunne opnast. Ventilane skal vere normally closed(NC). Dersom det skulle skje noko uventa som ein lekkasje vil det vere ynskjeleg å kutte tilførselen. Dei fleste ventilane i prosessen skal vere motoriserte. Desse ventilane treng ei viss tid for å opne/lukke og dette vil bidra til å redusere tilbakeslag som kan oppstå i røyra. Vi har intrigert nokre handstyrde ventilar og desse skal avgrense gjennomstrauminga av væske.

Krav

Når gruppa skulle finne ein motorisert ventil var det viktig at ventilen hadde høve til å gi tilbakemelding om posisjonen sin. Ved å få tilbakemelding om ventilane er opne eller lukka vil gi gruppa høve for å sikre mot at pumpene ikkje startar før ventilane er opna. Det vil òg gjer det mogleg å gi brukaren tilbakemelding om kva ventil som ikkje virke. I den manuelle styringa har dette blitt sikra mot ved hjelp av kopling. Ventilen må kunne opnast manuelt slik at ikkje bryggeprosessen stoppar opp dersom ein ventil skulle bli øydelagt. Sist men ikkje minst må ventilane halde den posisjonen som dei er i ved straumbrot. Det vil gjer det enklare for brukaren å gå over frå den automatiserte styringa til den manuelle styringa.

Kvifor CR5 01

Den motoriserte ventilen som skal nyttast i dette prosjektet oppfyller alle krava gruppa hadde sett. I tillegg må ein snu polane på ventilen ved opning og lukking. Dette gjer at ventilen vil halde den posisjonen som den har ved eit eventuelt straumbrot[8]. Ventilen har ikkje eit databladet, men på sida der ventilane vart kjøpt bruker ventilane mellom 3 til 5 sekund for å gå frå fult open til heilt lukka. Når gruppa testa ventilen brukte den 4,5 sekund. Den gir ein lyd frå seg når den byte posisjon noko som kan gi brukaren ein tryggleik ved at noko nytt held på å skje i prosessen.

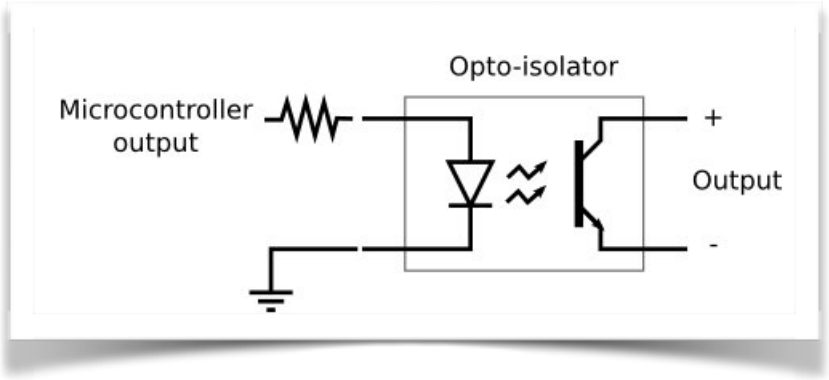
Figur 17 viser korleis ein kan manuelt opne ventilen om det eventuelt skulle skje ein straumfeil med ventilen.


Figur 17 Manuell styring av ventil

3.2.4 SSR

SSR står for “Solid State Relay” og er som namnet tilseier ei form for rørslefri relé. Desse verkar i prinsipp på same måte som eit konvensjonelt relé. Spenning blir sett på styringssida og releet lukker for å sleppe straum gjennom kontaktsettet. Det som skil SSRen frå tradisjonelle relé er at dei har ei svært rask opne-/lukketid (<=10ms)[9]. Det finnes ei rekke ulike typar SSR og det som er spesielt med dei vi skal bruke er at dei er laga for “switching” av høgare effektar enn andre SSR, ofte kalla triac. Releet er som oftast galvanisk skilt, som vil sei at styringssida ikkje er fysisk i kontakt med power-sida, og for å skilje desse er det ofte nytta ein optocoupler.

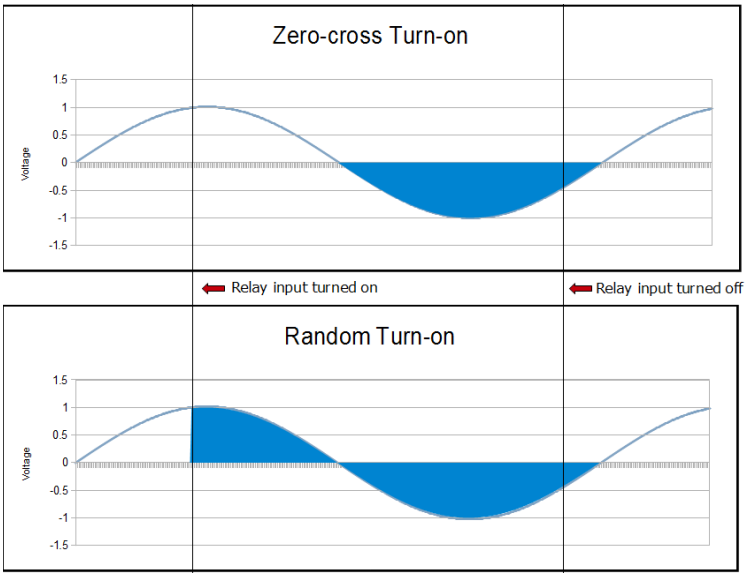
Optocoupler er ei transistorkopling som nyttar fototransistorar[10]. Styringssida styrer kun eit LED-lys som igjen styrer fototransistoren. Med ei slik kopling sikrar ein at høg straum på styringssida ikkje finn vegen til styringsorganet og øydelegg dette.



Figur 18 Optocoupler

Av di SSRane er bygd opp av halvleiarar har den ikkje noko bevegelege delar. Dette gjenspeglar den i si lange levetid og høge pålitelegheit. SSRen er mykje mindre enn eit elektromagnetisk relé, men av di den produserer mykje varme treng dei store kjøleribber som veg opp mot den “sparte” plassen. Den høge varmeutviklinga er ei av dei negative sidene ved desse relea. Utan tilstrekkeleg med kjøling vil SSRane fort bli øydelagt eller i verste fall ta fyr. Ei anna negativ side er at dersom SSRen blir øydelagt svikter den i lukka tilstand, noko som er ein potensielt farleg situasjon[11]. Vi har då valt å sette inn eit elektromagnetisk relé som kan kutte straumen dersom dette skulle skje.

Pulsering av høge straumar fører til store elektromagnetiske forstyrringar. Av denne grunn nyttar mange SSRar seg av “zero-crossover switching”, som går ut på at releet ikkje opnar/lukker seg midt i ei sinuskurve, men held posisjonen inntil straumen kjem under ein viss terskel. På denne måten unngår ein at desse høge forstyrringane blir generert[12].



Figur 19 Inn-/utkopling SSR

SSR RS1A

Då vi skulle velje SSRar hadde vi ikkje noko spesielle krav anna enn at dei måtte klare å handtere den straumen vi ville forsyne varmeelementa med samt at relea må kunne bli styrt av både RPi og ein PID. Vi var litt urolege for at desse relea ville medføra store forstyrringar på resten av utstyret, men sidan det var vanleg for desse relea å ha "zero-crossover switching" vart ikkje dette noko problem.

Vi kom over ein produsent frå USA som laga SSRar med dei spesifikasjonane som var viktige for oss. Desse relea hadde ein styringsspenning på 3-32VDC som gjer at vi kan styre dei direkte frå frå PIDen (12V) og RPi (5V). Produsenten kunne òg levere kjøleribber med tilstrekkeleg kjøling.

Vi var innom fleire idear når det gjeld å få tilstrekkeleg kjøling på SSRane. Vasskjøling henta frå ein PC, vifte/filter eller bruk av kjøleribbe på utsida av skapet til å trekke ut varmen, var nokre av ideane. Ved tilgang på tilpassa kjøleribber vart dette problemet løyst. Vi har likevel valt å bruke vifte/filter i skapet for ekstra kjøling.

Figur 20 SSR

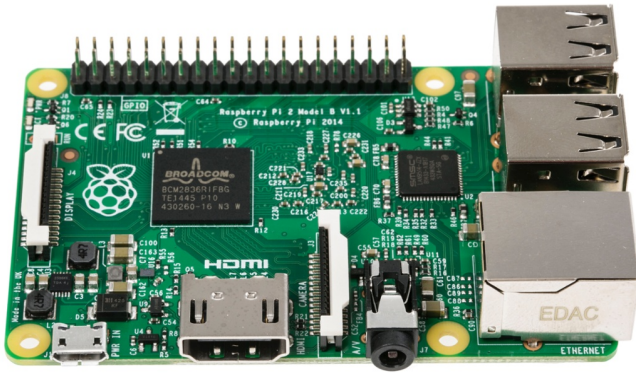
Valet fall tilslutt på eit relé på 25A for varmeelementet på 2500W og eit relè på 40A for varmeelementet på 5000W. Desse bestilte vi med tilhøyrande kjøleribber.

Implementering

Som nemnt opnar/lukker SSRen i nullfasen på sinuskurva. Det vil sei at den kan skifte tilstand opptil 120 ganger i sekundet. Vi valde ei sampletid på 200ms i programmet og i løpet av denne tida ville releet skifte tilstand to gongar, frå av til på til av. Med dette tidsintervallet er maksimalt tidsavvik på 8,3ms for kvar tilstandsendring, som utgjer 4,2% av sampletida. SSRen blir styrt av SSR-klassen som brukar 0-100% for å avgjere kor lenge releet skal vere av/på. Til naud vil det samla tidsavviket utgjer 8,4% avvik i denne reguleringa. Sjølv om dette avviket kan virke stort kjem avviket av kor i fasen releet blir aktivert. Dette betyr at avviket vil forskyva seg alt frå 0-8,2% som gir eit gjennomsnittleg avvik på 4,2%, som verkar meir overkommeleg. Dette avviket vil vere relativt konstant så PID-kontrollaren vil kunne regulere seg inn på dette. Grunnen til at vi valde ei så låg sampletid er av di lange intervall kan resultere i at SSRen vert øydelagt. Dette er noko vi blei gjort merksemd på då vi såg ein youtube-video om SSR[13].

3.2.5 Raspberry Pi 2 Model B

Raspberry Pi (RPi) er ein liten datamaskin på storleik med eit kredittkort. Den vart i hovudsak utvikla for å gi barn tilgang til ein lågpris data der dei lett kunne læra seg grunnleggande datakunnskapar. Gründeren bak dette prosjektet følte at barn ikkje tillærte seg desse kunnskapane lenger, slik som tidlegare generasjonar måtte. Dette er nok skuld i at utviklinga i dataverda har ført til at ein ikkje lenger treng å kode før ein brukar ein datamaskin. Utviklinga av RPi starta i 2006 og gjennom utviklingsprosessen har det skjedd ei stor utvikling i mobile chipar, som gjorde at dei kunne implementera fleire grafiske element. Då den fyrste RPien blei lansert var mottakinga eksepsjonell og innan to år hadde dei seld to millionar eksemplar[14].



Figur 21 Raspberry Pi

Tabell 1 viser spesifikasjonar for RPi.

	Raspberry Pi 2	Raspberry Pi B+
CPU	900 MHz quad-core ARM Cortex-A7	700 MHz single-core ARM1176JZF-S
GPU	Broadcom VideoCore IV @ 250 MHz	Broadcom VideoCore IV @ 250 MHz
RAM	1 GB (shared with GPU)	512 MB (shared with GPU)
Ports	4 USB, HDMI, 3,5mm Jack w/video, 10/100 Mbit/s Ethernet	4 USB, HDMI, 3,5mm Jack w/video, 10/100 Mbit/s Ethernet
I/O	16× GPIO plus the following, which can also be used as GPIO: UART, I²C bus, SPI bus with two chip selects, I²S audio +3.3 V, +5 V, ground.	16× GPIO plus the following, which can also be used as GPIO: UART, I²C bus, SPI bus with two chip selects, I²S audio +3.3 V, +5 V, ground.
Strømforbruk	800 mA (4.0 W)	600 mA (3.0 W)

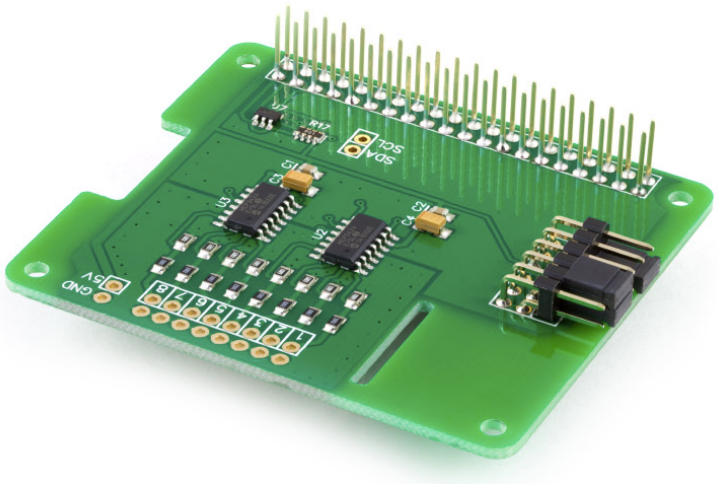
Tabell 1 Spesifikasjonar for Raspberry Pi

Kvifor Raspberry Pi

Då vi skulle starte prosjektet var val av styringsorgan veldig sentralt. På grunn av avgrensa ressursar var bruk av PLS med separate kort uaktuelt. Det vi trengte var eit allsidig kort som kunne takla både dei logiske - og grafiske elementa. Kommunikasjon mellom styringsorgan og brukaren måtte òg vere fleksibelt for å kunne takle eventuelle forandringar. Ved starten av prosjektarbeidet fann vi fyrst ein model, Raspberry Pi B+, men i løpet av prosjektperioden blei det lansert ein ny og utbetra modell, Raspberry Pi Model 2. Dennne modellen var seks gongar raskare enn den andre og sidan den nye modellen ville vere kompatibel med systemet vi hadde planlagt var det ingen problem for oss å gå over til denne.

3.2.6 Utvidingsmodular

Vi fann tidleg ut at Raspberry Pi 2 Model B (RPi) sin interne GPIO ikkje ville vere nok og at vi ville trenge utvidingskort. Vi søkte litt rundt på nettet og fann ei side der dei laga og selde kort som ville passe til vårt behov. Desse korta var “stackable”, noko som betyr at vi kan montere fleire av desse korta lagvis direkte på RPi´en. Dette sparer viktig plass og gjer at det enkelt òg kan supplerast med fleire kort om det skulle vere behov.



Figur 22 ADC Pi Plus

Ei utfordring med desse korta var at produsenten ikkje hadde noko bibliotek til Java. Vi sende ein epost til produsenten for å forhøyre oss om dette problemet. Svaret vi fekk tilbake frå produsenten AB Electronics UK kan du lese nedanfor. Fullstendig dialog mellom oss og produsenten kan du lese i vedlegg 8.

“*Thank you for your email. We have started working on a java lib using a third party library from http://pi4j.com/ which allows us to access the I2C ports using Java. At the moment the I2C support on their library is very limited and it is causing problems with our code when trying to read more than one byte at a time (the adc pi needs 4 bytes read in a block to get the value) and so until they have fixed this issue we are unable to proceed.*”

Sidan det ikkje var noko tilgjengeleg løysing enda måtte vi finne ei reserveløysing. Vi undersøkte og testa litt med å køyre Python-program i Java, men følte at vi måtte få teste dette direkte på modulen før vi kunne avgjere om det fungerte slik vi ville. Vi bestilte eit kort for å teste og det såg ut til at modulen klarte å lese verdiar kjapt nok for våre krav til systemet, så vi bestemte oss for å gå for desse korta.

I²C

Utvidingsmodulane nyttar I²C som kommunikasjonsbuss. I²C er ein seriell halvt duplex protokoll laga av Philips Semiconductor i 1982[15], for kommunikasjon mellom mange komponentar på eitt kretskort. I²C er ein lettvekt buss som kun nyttar seg av to leiarar. Kommunikasjonen føregår som følgjande:

1. Masteren etablerer ein start-tilstand. Dette gjer at alle slavane på linja venter og høyrer på instruksjonar.
2. Masteren sender adressa og ein lese/skrive “flag” ut på linja.
3. Slaven med den gjevne adressa sender så ut eit bekreftelsessignal
4. Kommunikasjonen kan no starta på databussen. Både masteren og slaven kan no lese/skrive alt etter om “flag” var lese eller skrive. Det blir sendt 8 bit med data kor mottakaren må bekrefta desse med eit 1 bits bekreftelsessignal.
5. Når kommunikasjonen er ferdig sender masteren eit stoppsignal og prosessen kan starta om igjen.

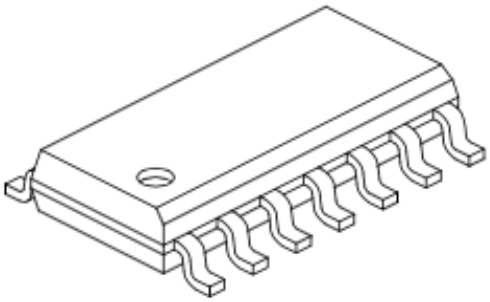
Figur 23 I²C-prosedyre

ADC Pi Plus

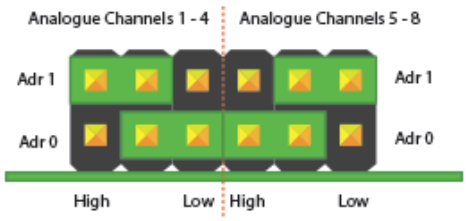
Dette kortet er laga for RPi modellane og har eit adressefelt som gjer det mogleg å plassere fire av desse korta oppå kvarandre, noko som gjer det lett å supplere med fleire kort ved behov.

Kortet er bygd opp av to MCP3424 IC frå Microchip Technologies Inc. IC`ane er 18 bits fire kanals analog til digital konverterar med ein del viktige tilleggsfunksjonar[16,17]. Det er ei innebygd programmerbar forsterking (x2, x4, x8) som forsterkar signalet før sjølve analog til digital omdanninga, noko som gjer at kortet kan operera med låge signal og samstundes gi ut nøyaktig data. Sample-tida kan òg bli satt i programmet, men raskare sample-tid går ut over oppløysinga (3,75SPS (18 bits) - 240SPS (12 bits)). Andre attributter er at chip`ane er svært driftseffektive med eit typisk forbruk på 36μA (135μA ved kontinuerleg konvertering), dei har eigen referansespenning og er sjølvkalibrerande. AB Electronics UK har tatt med alle desse attributta i sine kort og gjort dei tilgjengeleg i sitt Python-bibliotek.

Ein vel adresse på kortet via “jumpers” vist på figur 25. Figuren viser at det blir valt ei adresse for kvar chip.



Figur 24 Microchip



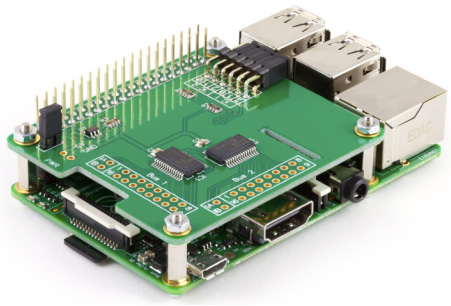
Figur 25 Adresselaskar for ADC Pi Plus

Konfigurering ADC Pi Plus

Vi valde å nytte oss av 15 målingar per sekund som gir oss ei oppløysing på 16 bits (0,076mV ved eit span på 5V). Då er vi godt innanfor dei krava vi har til målingane.

IO Pi Plus

Denne modellen nyttar seg av to chip`ar frå Microchip Technologies Inc av typen MCP23017, som er ein 16 bits I/O utvidingsmodul[18]. Kortet har innebygd 100KΩ “pull up” motstandar som kan bli konfigurert gjennom Python. Likeins med ADC Pi Plus er det sett av nok adressefelt til å kunne nytte fire av desse korta og adressa blir valt ved bruk av “jumpers”. Korta har høve til å hente ut straum direkte frå RPi eller å få dette supplert eksternt. Maksimalt straumuttak for kvar port (2 portar per kort) er 125mA og maks 25mA per I/O.



Figur 26 IO Pi Plus

Konfigurering IO Pi Plus

Vi har valt å nytte oss av to kort, eit for inngangar og eit for utgangar. Ved å eksternt supplere utgangskortet slepp vi å trekke all straumen gjennom RPi, noko som ville medført unaudsynt effekttap. Til inngangskortet har vi valt å nytte oss “pull up” motstandane internt på kortet. Dette kom vi fram til då testing viste at kortet var noko ustabilt utan “pull up/ down” motstandar. For å unngå lodding av “pull down” motastandar på alle inngangane gjorde vi heller nokre endringar på koplingsteikningane og la com`en på ventil-feedbacken til GND istaden for til Vcc som originalt planlagt.

3.2.7 LucidControl RT4

Problemstilling

Temperaturregulering har blitt ein sentral og viktig del av den nye bryggar sitt bryggesystem. Under heile bryggeprosessen frå start til slutt er det krav om ulike temperaturar. Ved mesking er det satt ein mesketemperatur, mens i nokre oppskrifter skal ein meske i fleire ulike temperaturar, òg kalla trinnmesking. Før ein startar å koke må ein skylle gjennom kornet, dvs tilsetje ein del vatn til brygget. Dette må tilfredsstilla ein vis temperatur og ved koking må temperaturen på brygget nå ein koketemperatur. Til slutt skal brygget kjølast før det skal tappast over på gjæringskar, kor gjæren har eit temperaturområde som må tilfredsstillas for at gjæren skal klare å jobbe.

Figur 27 LucidControl RT4

Kvifor LucidControl RT4

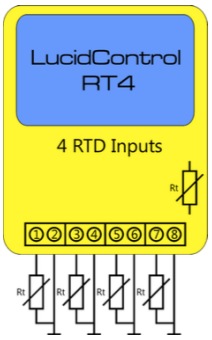
Til prosjektet vil vi trenge fire temperaturmålarar som dekker eit måleområde frå 0 til 120ºC. Når vi skulle finne ein metode for korleis vi skulle måle temperaturar falt vi raskt for eit Pt100 element. Pt100 element kjem inn under kategorien RTD-element som er av typen platina og har ein indre motstand som endrar seg med temperaturen. Gruppa har ein del kjennskap til desse elementa frå tidlegare. Dette er element som vert mykje nytta av industrien[19].

Elementa skal kunne lesast av eit Raspberry Pi (heretter RPi) inngangskort. Inngangskortet les spenningar fra 0 - 5V. Dette gjer at vi treng fire måleomformarar for å ha moglegheit til å hente inn temperaturmålingane. I søket etter desse omformarane kom gruppa over ei heimeside som selde plug-and-play USB modular[20]. Ein av deira modular ved namn «LucidControl RT4 - 4 Channel RTD Input USB Module» hadde moglegheit til å ta imot 4 Pt100/Pt1000 element. Modulen kan koplast direkte til ein RPI med ein USB-kabel. Gruppa velde til slutt å gå for denne løysinga og slepp då å prosjektere med fire omformarar.

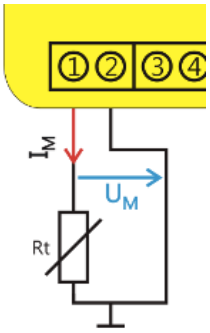
Gruppa kontakta LucidControl med ein førespurnad om LucidControl RT4(heretter LCRT4) modulen. Førespurnaden og svaret kan du sjå i vedlegg 8. LCRT4 modulen støttar kun Pt1000 element, så vi velde å gå over til desse elementa og implementera LCRT4 modulen til systemet.

Eigenskapar

Modulen har eit måleområde frå -180 ºC til 180ºC med ein målegrannsemd på +/- 0.5ºC. Den kan koplast direkte til operativsystem som Windows, Linux, RPi og Beagle Bone. Modulen har fleire drivarar, java-bibliotek tilgjengeleg, den kan kontrollere om temperaturelementet er i orden og om det er brot i kretsen.



Figur 28 RTD Inputs

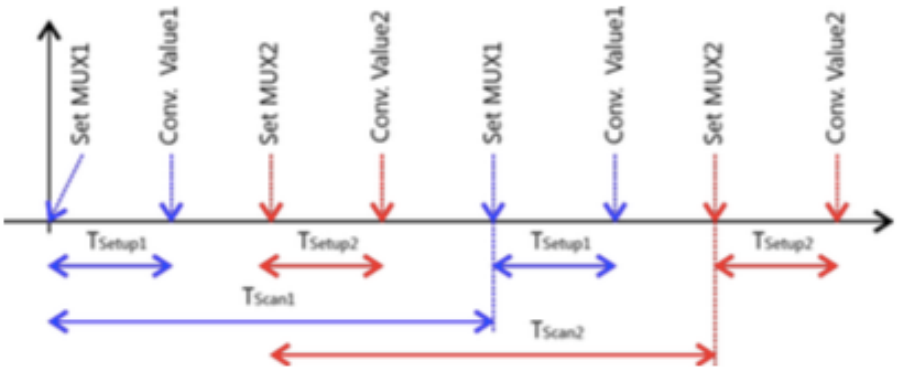


Figur 29 Tilkopling PT-1000

Behandling av data

Når ein temperaturmålar vert kopla til ein LCRT4 modul og kretsen slutta (figur 29), vil det bli generert ein straum. Temperaturmålaren har ein indre motstand som endrar seg ved temperaturendringar. Når temperaturen aukar så vil den indre motstanden til temperaturmålaren auke parallelt, samstundes som spenninga over temperaturmålaren endrar seg. Dette utnytter LCRT4 seg av og ved hjelp av desse endringane har LCRT4 moglegheit til å rekne ut kva temperaturen er.

I databladet til LCRT4 er det lagt ut ei beskriving på korleis modulen opererer for å hente inn temperaturmålingane. Figur 30 på neste side er henta frå databladet og viser sekvensstyringa til to tilkopla temperaturmålarar. Modulen henter ikkje inn målingar frå alle temperaturelementa samstundes. Den jobbar sekvensielt gjennom dei sensorane som er tilkopla. La oss sjå på dømet frå databladet. Til kvar sensor er det tilknytta to tidsperiodar og to operasjonar. Tidsperiodane «Tsetup» og «TScan» har ein moglegheit til å endre på. «Tscan» avgjer kva tid kvar ny måling skal bli utført. «Tsetup» er tida kretsen treng for å stabiliserast slik at ein får nøyaktige verdiar. Operasjonane som vert utført er «Set MUX» og «Conv. Value». «Set MUX» er når kretsen til sensoren blir straumsatt og «Conv. Value» er når modulen konverterar målingane til ein temperaturverdi som så kan lesast via USB-kabelen.



Figur 30 Sekvensstyring

Testing

Når vi skulle teste LCRT4 kopla vi den til RPI. Vi lasta ned «LucidIoCtrol Command Line Tool»[21] og flytta mappa over på RPI. Forbindinga mellom gruppa sine datamaskinar og RPi var etablert på førehand. Vi opna terminalvindauget og kopla oss til RPi's kommandovindauge. Deretter opna vi mappa og køyrde scriptet direkte i kommandovindauget. Scriptet opna då forbinding mellom RPI og LCRT4 via USB-kabelen. Temperaturmålingane på dei fire ulike kanalane vert henta og vist, vist i figur 31.

```
pi@raspberrypi ~ $ cd LucidIoCtrl_1_2_1_RPi/
pi@raspberrypi ~/LucidIoCtrl_1_2_1_RPi $ ./LucidIoCtrl -d/dev/ttyACM0 -tT -c0,1,2,3 -r
CH0:ERR_OPEN    CH1:ERR_OPEN    CH2:ERR_OPEN    CH3:ERR_OPEN
```

Figur 31 Kode for å hente temperatur

Koden «ERR_OPEN» visar oss at det no ikkje er noko tilkopla LCRT4. Det fyrste vi gjorde var å finne ein Pt1000-tabell som visar kva motstand den har i ulike temperaturar[22]. Vi kopla til fire ulike motstandarar som skulle simulera fire ulike temperaturmålarar. Vi hentar ut to verdiar frå tabellen, motstanden til element ved 0ºC og 100ºC.

Pt1000 ved 0ºC = 1000.0 ohm
Pt1000 ved 100ºC = 1385.1 ohm

No har vi moglegheit til å rekne oss fram til kor mykje motstanden til Pt1000 elementet aukar med for kvar grad. Figur 32 på neste side viser korleis gruppa løyste dette. Det fyrste vi gjorde var å finne ut kor mange ohm motstanden aukar med, frå 0ºC til 100ºC. Deretter trakk vi motstanden ved 0ºC frå motstanden ved 100ºC og delte verdien på det totale spranget frå 0ºC til 100ºC, som er 100.

$$\frac{Pt1000(100^{o}C) - Pt1000(0^{o}C)}{100} = \frac{1385.1\Omega - 1000.0\Omega}{100} = 3.851\Omega$$

Figur 32 Utrekning av Ohm/˚C

Resultatet vart at motstanden aukar med 3.851 ohm ved ei grads auke. Vi kan no rekne oss fram til kva temperaturar motstandane skal simulera ved å endre på formelen vi nytta i figur 32.

$$Motstand\ 1 = \frac{657\Omega - 1000\Omega}{3.851\Omega} \approx \underline{-89.07^{o}C}$$

$$Motstand\ 2 = \frac{1196\Omega - 1000.0\Omega}{3.851\Omega} \approx \underline{50.9^{o}C}$$

$$Motstand\ 3 = \frac{1491\Omega - 1000.0\Omega}{3.851\Omega} \approx \underline{127.5^{o}C}$$

$$Motstand\ 4 = \frac{1490\Omega - 1000.0\Omega}{3.851\Omega} \approx \underline{127.2^{o}C}$$

Figur 33 Utrekning av temperatur

Vi har no rekna ut kva temperatur motstandane skal representera og køyrer scriptet på nytt, vist i figur 34.

```
pi@raspberrypi ~ $ cd LucidIoCtrl_1_2_1_RPi/
pi@raspberrypi ~/LucidIoCtrl_1_2_1_RPi $ ./LucidIoCtrl -d/dev/ttyACM0 -tT -c0,1,2,3 -r
CH0:-86.050     CH1:50.940      CH2:128.370     CH3:128.500
```

Figur 34 Kode for å hente temperatur

Etter at vi hadde køyrd scriptet sette vi alle verdiane inn i ein tabell for å få ei oversikt.

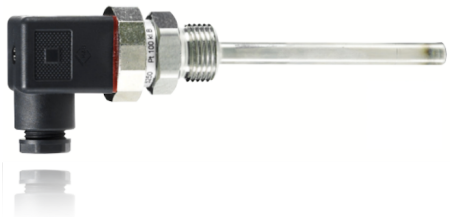
	Channel 0	Channel 1	Channel 2	Channel 3
Motstand	657 ohm	1196 ohm	1491 ohm	1490 ohm
Utreknet Temp.	-89.07 ºC	50.9 ºC	127.5 ºC	127.2 ºC
Målinger	-86.05 ºC	50.94 ºC	128.37 ºC	128.5 ºC
Avvik	**3.02 ºC**	**0.04 ºC**	**0.87 ºC**	**1.3 ºC**

Tabell 2 Verdiar henta frå script

Hensikta med testen var å sjekke om modulen verka og om vi fekk kontakt med modulen. Dette var ikkje ein test på komponenten si grannsemd. Det avviket som har oppstått kan vere eit resultat av fleire ting. Til dømes avlesing av multimeter som vart nytta for å ohm'e motstandane og avrundingar vi gjorde under utrekning.

3.2.8 PT-1000

Prosjektet er no prosjektert med PT-1000 element. Det neste vi gjorde var å finne PT-1000 element som kunne passe til prosjektet vårt. Som nemnt tidlegare hadde gruppa satt eit måleområde som sensoren måtte dekka, frå 0ºC til 120ºC. Vi kom fort over nokre PT-1000 element som Danfoss selde[23]. Gruppa velde til slutt å gå for sensoren av typen MBT 5250 som har eit måleområde frå -50°C til 200°C. Sensoren er kapsla inn for beskyttelse slik at den ikkje kjem i direkte kontakt med det den skal måle. Dette gir oss moglegheit til å skru sensorane direkte inn på gryta der dei skal nyttast.



Figur 35 PT-1000

3.2.9 DP-celle

Kvifor DP-celle

Ute i bryggeprosessen har gruppa prosjektert med tre nivåmålarar. Nivået skal målast i tre forskjellige tankar. På markedet finst det utruleg mange løysingar og utstyr for å integrere ein nivåmåling til ein prosess. Det er difor viktig å vurdere i kvart enkelt tilfelle kva løysing ein ynskjer å bruke. Gruppa sette seg difor ned og vurderte utfordringar som vil oppstå i det systemet som er prosjektert. Nedfor kan du sjå ei liste med utfordringane vi vurderte:

Figur 36 DP-celle

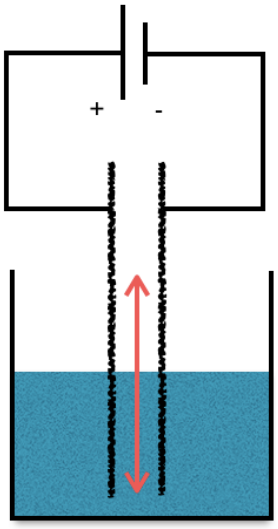
- Skumdanning ved koking
- Tilsetjingar
- Endring i massetettleik og ledeevne
- Temperaturendringar
- Damp
- Opne tankar
- Enkel fråkopling for reingjering
- Ulike dimensjonar på tankane
- Pris

Gruppa vurderte så tre ulike løysingar:

- Kapasitans
- Ultralyd
- Dp-celle

Kapasitive målar

Denne målemetoden bygger på kondensator prinsippet. Ein kondensator er satt saman av to plater som er skilt frå kvarandre. Nokre kondensatorar har òg eit stoff mellom platene kalla dielektrikum[24]. Enkelt forklart så har kondensatoren moglegheit til å lade seg opp. Kor mykje den klarer å lade seg opp blir bestemt av kondensatorens kapasitans, formelen finn du i figur 37.



Figur 38 Kapasitiv målar

$$C = \varepsilon \frac{A}{l} \quad [F]$$

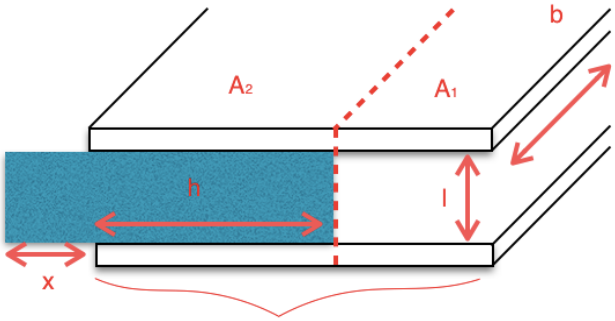
Figur 37 Formel

C - *kapasitansen*

ε - *dielektritetkonstanten*

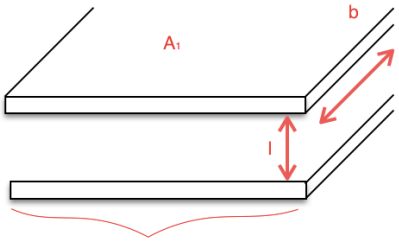
A - *Areal*

l - *avstanden mellom platene*



Figur 39 Posisjonsmålar

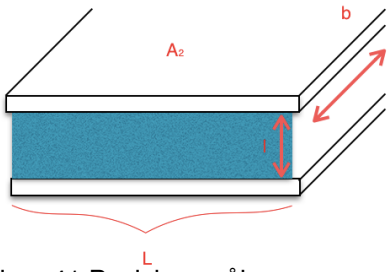
Det kapasitive måleprinsippet fungerer på nesten akkurat same måte. For å gjere det enklare å forklare har gruppa laga tre figurar. Figurane representerer tre ulike tilfeller som kan oppstå i måleprosessen. Figur 40 viser når tanken er tom og figur 41 viser når tanken er full. Arealet til platene og avstanden mellom dei er konstant. Det som endrer seg er det som er mellom platene, dielektrisitetskonstanten. Med denne informasjonen kan vi setje opp ein ny formel, figur 42. I den nye formelen kan vi sjå at det er to ledd. Det fyrste leddet representerar tanken når den er tom, figur 40, det andre leddet er når tanken er full, figur 41.



Figur 40 Posisjonsmålar

Vi har no moglegheit til å finne måleområdet til sensoren. Når nivået i tanken endrar seg vil òg kapasitansen endre seg parallelt. Når vi veit kapasitansen kan vi snu på formelen og

finne variabelen h i figur 42, som representerar høgda på nivået i tanken.



Figur 41 Posisjonsmåler

1) $$C = \frac{\varepsilon_0 A_1}{l} + \frac{\varepsilon_0 \varepsilon_r A_2}{l}$$

2) $$C = \frac{\varepsilon_0}{l}(A_1 + \varepsilon_r A_2)$$

3) $$\frac{Cl}{\varepsilon_0} = (A_1 + \varepsilon_r A_2) \qquad A_1 = (L - h)b \qquad A_2 = \varepsilon_r hb$$

4) $$\frac{Cl}{\varepsilon_0} = (L - h)b + \varepsilon_r hb$$

5) $$\frac{Cl}{\varepsilon_0 b} - L = \varepsilon_r h - h$$

6) $$\frac{Cl}{\varepsilon_0 b} - L = h\,(\varepsilon_r - 1)$$

7) $$h = \frac{Cl}{\varepsilon_0 b(\varepsilon_r - 1)} - \frac{L}{(\varepsilon_r - 1)}$$

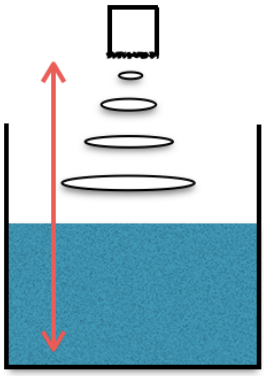
Figur: 42 Utrekning

Fordelen med dette måleprinsippet er at den ikkje blir påverka av skum og temperaturendringar.

Ulempa ved å bruke ein kapasitiv målar er at den er i direkte kontakt med mediumet. Den må monterast i toppen av gryta.

Ultralyd

Kort forklart fungerer ultralydsensoren slik at den sender ut ein ultralydpuls ned i tanken. Når pulsen treffer overgangen mellom luft og væske, vil ein del av pulsen reflekterast tilbake til følaren. Tida ultralydpulsen bruker frå den blir sendt til den så kjem tilbake er eit mål på nivået i tanken[25].



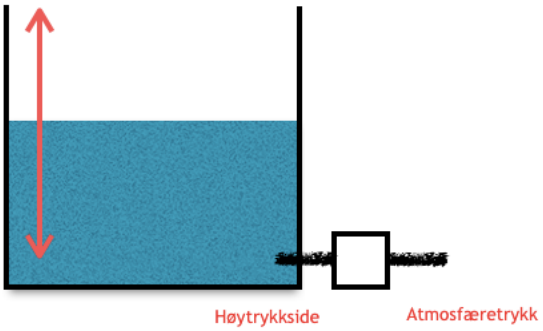
Figur 43 Ultralydmåling

Fordelen ved å nytte ein ultralydsensor er at den ikkje er i direkte kontakt med væska som skal målast. Dette gjer at den ikkje vert påverka av væska si massetettleik og ledeevne.

Ulempene ved å nytte ein ultralydsensor er at den må monterast over grytene. Dette kan gjere det innfløkt og kan påverke målingane om ein skulle vera uheldig å dekkja til sensoren ved ein eventuell tilsetjing. Ultralydpulsane vert òg påverka av temperaturendringar. Det er sjølve lydhastigheten som endrar seg ved temperaturendringar[26]. Prisklassen til ein ultralydsensor er brei, ein kan finne både billige og dyre.

Om DP-celle

DP-cella bygger på eit prinsipp kor vi måler skilnaden mellom to trykk. DP-cella er bygd opp av to trykkammer som er delt i midten av eit membran. Vi får ei høgtrykkside som skal koblast til tanken og ei lavtrykkside som har fri tilførsel til atmosfæretrykk. Det er viktig under montering at du kopler rett, ellers kan du øydeleggje DP-cella.



Figur 44 Trykkmåling

Det er òg viktig å tenke på kor du plasserer DP-cella. Dette av di DP-cella tar utgangspunkt i kor den vert kopla til. Kopler ein den over eller under botnen av gryta må dette bli tatt med i utrekningane.

Formelen for differansetrykket over ei Virke-celle er vist i figur 45.

$$P = h \cdot \rho \cdot g$$

Figur 45 Formel

h - *høyden*

g - *tyngdekraft*

ρ - *massetettheten*

Fordelen ved å bruke ei DP-celle er at den kan monterast på undersida av gryta, noko som gjer at målingane vert gjort frå undersida og ikkje ovan ifrå. Dette gjer at målingane ikkje vert påverka ved skumdanningar. Vidare påverkar ikkje temperaturendringane noko på måleresultatet. Dei er òg enkle å kople frå for vedlikehald. Det breie utvalget av DP-celler på marknaden gjer at ein ikkje treng å betale for mykje for ein fult fungerande DP-celle.

Ulempa ved å bruke ei DP-celle er at den er i direkte kontakt med væska og ikkje tar omsyn til at massetettleiken i væska endrar seg. Vi må bruke eit filter for at DP-cella ikkje skal køyre seg tette av ulike tilsetjingsstoff.

Konklusjon

Måleprinsippa har alle sine fordelar og ulemper. Når gruppa til slutt skulle velja eit av måleprinsippa, gjekk vi for DP-cella. Den er enkel å bruke og ein kan få dei til ein rimeleg pris. Dei er òg enkle å kople frå slik at ein har moglegheit for å reingjere utstyret. DP-cella som gruppa har kjøpt inn og som er brukt på prøvestasjonen/prototypen er ikkje den som vil bli brukt på sjølve prosjektet. Desse vart kjøpt inn for at gruppa skulle ha moglegheit for å teste programmet, men funksjonaliteten vil vera den same.

Kort om MPX5010

DP-cella gruppa har nytta er lita og kompakt, figur 36. Nivåmålaren har moglegheit til å måle trykk frå 0 til 10kPa[27]. Ved å snu på formelen i figur 45 har vi moglegheit for å finne ut kor høgt dette tilsvarar. Utrekningane er gjort med massetettleiken til vatn og resultatet kan du sjå i figur 46. Med denne DP-cella har vi moglegheit til å måle ei vassøyle på ca 1m. DP-cella gjer om trykkverdiane til ei spenning frå 0.2V til 4,7V. Spenningsverdien blir lest av eit analogt inngangskort på Raspberry PI.

$$h = \frac{P}{\rho \cdot g} = \frac{10kPa}{1000kg/m^3 \cdot 9.81m/s^2} = 1.02m$$

Figur 46 Utrekning av vassøyla

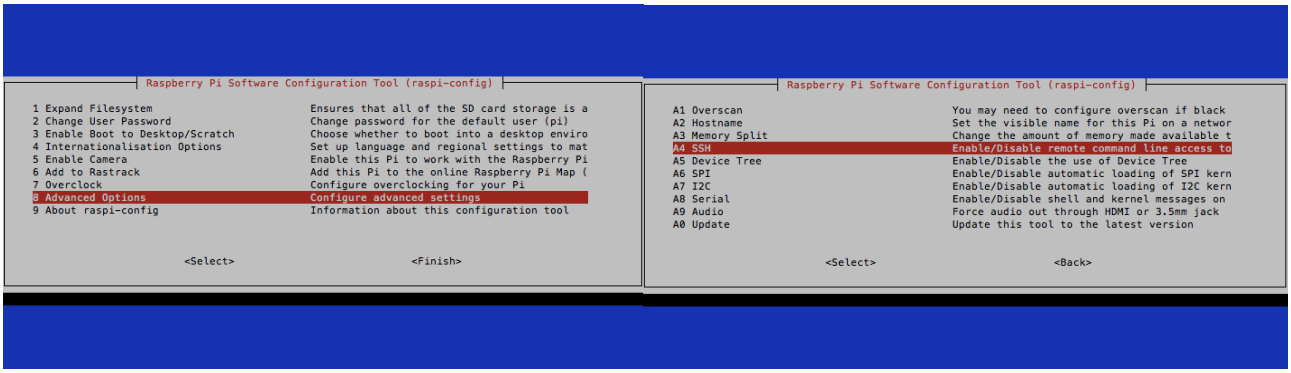
3.3 Programvare

Gruppa ville etablera ein komplett programvarepakke slik at alt frå planlegging til utføring av eit brygg skulle vere enkelt og intuitivt. Det vart laga to program til å oppnå dette med og ein database som skulle ta seg av problemfri synkronisering. I RecipeMaker kan bryggar legge til oppskrifter, kva tid som helst og frå kor som helst, så lenge bryggar har tilgang til PC. Når oppskriftene er lagra kan bryggaren gå over til BrewingScreen, finne sine oppskrifter og starte brygginga.

3.3.1 Konfigurering av Raspberry Pi

Det fyrste vi måtte gjere var å installere operativsystemet på eit SD-kort. Vi valde å bruka eit OS kalla Raspbian. Raspbian er ein gratis Debian(Linux) distribusjon utvikla spesifikt til Raspberry hardware og inneheld over 35 tusen ferdigkompilerte pakkar. Dette gjer at det er mykje godt støtte tilgjengeleg (m.a. touch-skjerm). Raspbian er ikkje utvikla av RPi-teamet men ei uavhengig gruppe dedikerte utviklarar[28].

RPi-nettsida “hoster” dei mest vanlege distribusjonane av OSar og ein guide til korleis installere desse til eit SD-kort[29,30]. Etter at vi hadde installert Raspbian måtte vi konfigurere nokre innstillingar for å få tilgang til ulike protokollar og for å betre ytinga. Dette blei gjort i det fyrste vindauget som kom opp etter installasjonen, som vist i figur 47.



Figur 47 Konfigurering

I desse vindauga aktiverte vi bruken av SSH (secure shell) og I2C. Sidan touch-skjermen ikkje har noko høgtalar tvang vi lydutgangen gjennom mini-jack`en. Her sette vi òg “memory split”. RPi deler ei avgrensa mengd med RAM for både CPU og GPU og må fordele dette minne mellom desse to. Vi gav GPU`en 256MB RAM. For å betre ytinga valte vi å “overclock`e” RPi med pre-sett Pi2.

SSH (Secure Shell)

Vi nytta SSH for å kunne styre RPi ved bruk av PCane våre. På denne måten kunne vi jobbe på RPien utan å sitje med den. SSH er ein nettverksprotokoll som nyttes for å få tilgang til kommandolinjer til ein annan maskin[31]. På MAC/Linux-maskiner kan ein enkelt kople seg til RPi ved hjelp av kommandoen “ssh pi@<IP>”, vist i linjene nedanfor.

```
Last login: Sun May 10 13:00:56 on console
Mikal-Berge-2:~ MikalBerge$ ssh pi@192.168.80.132
Linux raspberrypi 3.18.11-v7+ #777 SMP PREEMPT Sat Apr 11 17:30:37 BST 2015 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 11 22:01:37 2015 from 192.168.80.194
pi@raspberrypi ~ $
```

Som ein kan sjå i utskrifta vart vi ikkje spurt etter noko form for passord og dette er av di vi nytta noko kalla “SSH keys without passphrase”. For å generere denne nykelen følgde vi rettleiinga på RPi sine nettsider[36].

Vi måtte fyrst generere ein nykel til bruk i RPi.

```
Mikal-Berge-2:~ MikalBerge$ ssh-keygen -t rsa -C "pi@<RPi's IP
Address>"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/Mikal/.ssh/id_rsa): /Users/
Mikal/.ssh/id_rsaRapport
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/Mikal/.ssh/id_rsaRapport.
Your public key has been saved in /Users/Mikal/.ssh/id_rsaRapport.pub.
The key fingerprint is:
64:d6:04:8d:bf:ea:2d:5e:8e:1d:9a:bc:e4:92:81:e8 pi@<RPi's IP Address>
The key's randomart image is:
+--[ RSA 2048]----+
|        .+.      |
|        .o.      |
|        +..      |
|       +  .      |
|   . .  S  .     |
|  . . .   .      |
| .     o..o      |
```

Vi sjekka så om nykelen hadde blitt generert.

```
Mikal-Berge-2:~ MikalBerge$ ls ~/.ssh
id_rsa           id_rsaRapport        known_hosts
id_rsa.pub       id_rsaRapport.pub
```

```
cat ~/.ssh/id_rsaRapport.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDO
+iCKAwdyuRirW6F1V1UaDwkWqPS5OMuY9HIU4JeoEOMSlUAZEmbKq8UgZHSSm8lQf8aaYDP
QKLfzyL7XsU719MnoUPUP7qCgVZAC5Vg3S8q6yIFKNI6ztdDCbk84x+pcGu2m2dyW
+yB5F8AJieaM0Zu+DZt2wUVMNmjRIVFHEJpxWdth6LdwWspgZ
+Gxj291yxs0+qGRviWPCaVdM+t/jV0B/Cvm2qR3M3+mjE93ZcCb/
wlxBClefCLKevlWsDqJtrQYsmpQmBQP8l5Y2P9a+/ZJ
+BHNr1MHazY2z0is72TiVWr1OQTWDKy48qSDRYayUfL0eN8vivT/KT1cstCZ pi@<RPi's
```

id_rsaRapport er vår private nykel som må vere på vår maskin. id_rsaRapport.pub er nykelen som skal vere på maskinen vi vil kople oss til, slik at den veit at det er vi som prøver å kople oss til.

```
Mikal-Berge-2:~ MikalBerge$ cat ~/.ssh/id_rsa.pub | ssh
pi@192.168.XX.XXX 'cat >> .ssh/authorized_keys'
```

Kommandoen ovanfor kopierer .pub nykelen over til RPien. No kan vi kople oss til RPi via SSH utan å taste inn noko passord. SCP (Secure Copy) nytta vi til å sende over programma vi ville prøve å køyre.

```
Mikal-Berge-2:~ MikalBerge$ scp NetBeansProjects/AutoBrygg/
BrewingScreen/Jar/BrewingScreen-0.05-SNAPSHOT.jar pi@192.168.80.132:
BrewingScreen-0.05-SNAPSHOT.jar
```

pi4j
Etter at kommunikasjonen var etablert var det framleis nokre småting vi måtte ordne på. Fyrst måtte vi kontrollere at vi fekk kontakt mellom java og GPIO-ane. Tilgang til GPIO-ane kan vi få gjennom pi4j biblioteket. Vi installerte pi4j biblioteket med kommandoen `curl –s get.pi4j.com | sudo bash`
Etter at pi4j var installert kunne vi bruke den med kommandoen `pi4j` som fortel java at den skal inkludere pi4j biblioteket. pi4j er eit prosjekt for å knyte java til RPi sine I/O-ar. Velkomstteksten på nettsida skildrar det presist:

"*This project is intended to provide a friendly object-oriented I/O API and implementation libraries for Java Programmers to access the full I/O capabilities of the Raspberry Pi platform. This project abstracts the low-level native integration and interrupt monitoring to enable Java programmers to focus on implementing their application business logic.*" [33]

Testen vi utførte var ganske enkel. Vi ville teste om biblioteket var installert korrekt og om vi kunne bruke det. Vi laga eit enkelt oppsett med ein knapp og eit LED-lys som lyste når vi trykte på knappen. Dette var nok til å konstatere at programvara fungerte og at vi hadde tilgang til GPIO-ane.

Etter litt testing fann vi ut at vi ikkje trengte å bruke `pi4j` kommandoen dersom vi inkluderte biblioteket i .jar-fila. Vi kunne køyre den konvensjonelle kommandoen `java –jar */*.jar` .

Python
Neste ting på agendaen var å opprette kommunikasjon med I/O-utvidingsmodulane, som vi skulle prøve å køyre med Python istaden for Java. Dette av di det ikkje fantes noko bibliotek vi kunne implementere. Tanken var å bruke Java til å kalla på Python-programma og sette/lese frå utgangen.

Fyrst sjekka vi om vi fekk til dette direkte i Python, som allereie var installert på RPien. Vi lasta ned nokre eksempelprogram med kommandoen `git clone https://github.com/abelectronicsuk/ABElectronics_Python_Libraries.git`. Før vi kunne køyre desse programma måtte vi installere eit Python-bibliotek kalla “smbus”. Dette biblioteket gjer det mogleg for Python å lese/skrive til I2C-bussen. Vi lasta ned biblioteket med kommandoen `sudo apt-get install python-smbus`.

Neste steg var å sjå om vi kunne køyre desse klassane gjennom Java. Vi testa fyrst noko kalla Jython, som er eit bibliotek ein til dømes kan bruke til å hente ut/setje ein variabel i ein Python-klasse. Ved bruk av denne metoden støytte vi på nokre problem. Den var veldig treg og det oppstod problem med å importere bibliotek i Python.

Vi prøvde å køyre Python-programma med exec-metoden til runtime-klassen i Java. Runtime-metoden verka raskare og vi fekk ikkje problem med å importera klassar inn i

Python. Verdiar vi henta ut frå prosess-klassane i Java gjorde det mogleg å lese utgangen til Python-programmet og kaste dei til variablar i Java.

Touch-skjerm
Touch-skjermen har vi hatt nokre problem med. Den krevde ein drivar som ikkje var inkludert i OS-en (dette vart seinare støtta i ein ny versjon av firmware`en). Vi brukte mykje tid og mange forsøk på kompileringa av ein ny kernel for å opprette støtte for denne[34]. Vi fekk òg problem med å lage kalibreringa permanent. Då vi bytta frå RPi B+ til RPi 2 trengte vi ikkje gjere noko for å få kontakt med skjermen. No trengte vi berre å kalibrere skjermen.

Noko vi må sjå litt nærare på er å få desse kalibreringane til å virke i “FrameBuffer”. Dei kalibreringsprogramma vi fann, som virka, kalibrerte for X11. Dette ville ikkje ha noko å sei for programmet vårt (JavaFX-program blir køyrt direkte i FrameBuffer).

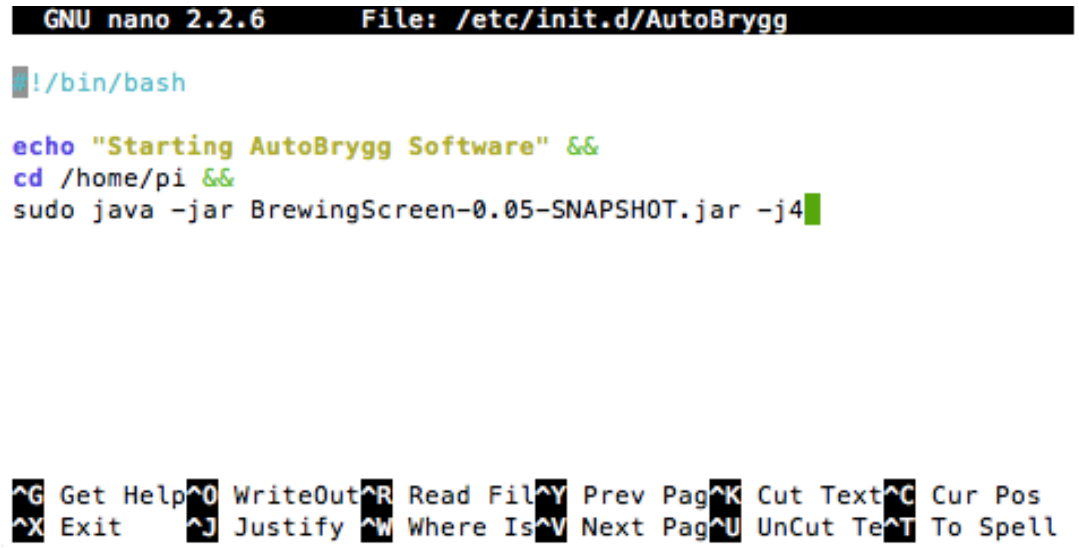
Statisk IP-adresse
Det var i utgangspunktet ikkje naudsynt å legge til IP-adresse, men ein dag fekk vi ikkje kontakt med RPIen og fann ut at den hadde fått utlevert ei ny IP-adresse. Vi ordna dette ved å legge til nettverksinformasjonen inn i fila `/etc/network/interfaces`.[34] Vi endra `iface eth0 inet dhcp` til `iface eth0 inet static` og måtte legge til følgjande informasjon:

address	192.168.XX.XXX	(RPi si noveradne IP-adresse)
netmask	255.255.255.0	(Nettet si nettmaske)
network	192.168.XX.X	(Ruteren si IP-adresse)
broadcast	192.168.XX.XXX	(Kringkastingsadressa)
gateway	192.168.XX.X	(Nettet sin gateway)

Boot script[35]
Vi ville lage eit script som blir køyrt ved oppstart av RPi som starta Java-programmet. På denne måten slepp brukar å køyre kommandoar i RPi sin terminal.

Vi starta med å lage scriptet `sudo nano /etc/init.d/AutoBrygg`.

Med kommandoen ovanfor blir det generert ei tom fil i `/etc/init.d/` -direktoratet med namn AutoBrygg. Denne fila er òg opna i eit tekstprogram, der vi la inn alle kommandoane vi ville køyre:



Figur 48 AutoBrygg script

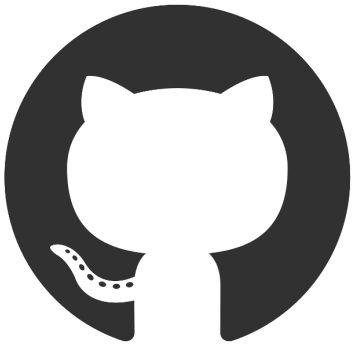
Etter at dette scriptet var laga måtte vi gi det nokre rettigheiter. Dette blir gjort med `chmod` kommandoen. I vårt tilfelle måtte scriptet få køyrbare rettigheiter (`chmod 755`) som vi løyste med denne linja: `sudo chmod 755 /etc/init.d/AutoBrygg`

Vi måtte òg sette at dette scriptet skal bli køyrt ved oppstart:
`sudo update-rc.d AutoBrygg defaults`

Scriptet kan bli fjerna frå oppstartslista med denne linja:
`sudo update-rc.d -f AutoBrygg remove`

3.3.2 GitHub

Om Git og GitHub



Figur 49 GitHub

Git er eit distribuert revisjonskontrollsystem til bruk på software. Dette vart utvikla av Linus Torvalds som 7. April 2005 lanserte sitt svar på revisjonskontroll. Systemet har blitt industristandarden til å spore utviklinga av eit prosjektet. I eit intervju seier Torvalds at dåtidas revisjonskontrollsystem virka dårleg dersom to greiner av programmet skulle settast saman (òg kalla merge). BitKeeper hadde tatt vekk si gratis støtte noko som gjer handteringa av Linux-kernel-utviklinga vanskeleg[36,37].

Grunna den dårlege merge-støtta førte dette til at mange brukarar av desse systema velde å ikkje nytta desse greienene. Dette førte til at fleire jobba direkte på ein kopi av hovuDProgrammet. Torvalds ville gjere merge´inga og avgreininga(branching) enklare for å gjenopne bruk av greiner, slik at brukarar kan ha fleire oppgåvededikterte greiner[36,38]. Git har lagt mykje vekt på å støtte non-lineær programutvikling og brukarar skal då ha høve for å jobbe på kvar si grein av programmet og merge dette med hovudgreina(master branch)[39].

Dei mest vanlege git-metodane er:

`commit`[40]
Commit er ein kommando som vert nytta for å lagre tilstanden til prosjektet i noverande tilstand, kalla “snapshot”. Etter commit kommandoen er nytta vil git spore nye endringar frå den nye tilstanden. Commit bør alltid innehalde ein commit-melding som fortel kva som har endra seg mellom commit’ane.

`push`[41]
Push er ein kommando som vert nytta for å “skyve” all commit-historikken og endringar i filene frå maskin til ein “remote”-git, til døme ein git som er på GitHub.

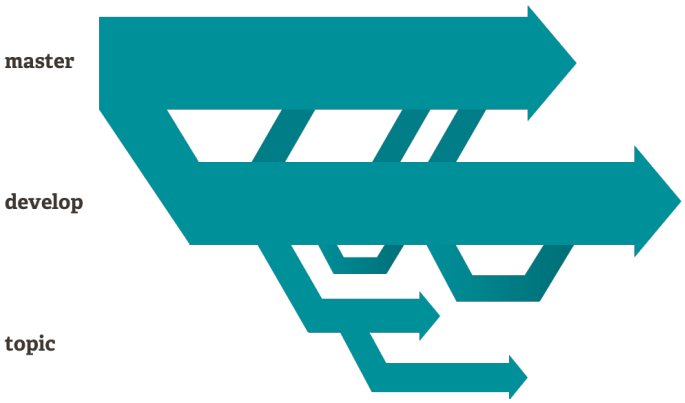
`pull`[42]
Pull vert òg nytta på remote-git, men gjer motsatt av push. Pull kommandoen henter git-versjonane som er lagra i remote og lastar det inn til din locale-git. Denne kommandoen vert oftast nytta dersom det er fleire personar som jobber på det same prosjektet og på denne måten kan brukarar synkronisere program.

`clone`[43]
Clone virker nesten på same måte som ein last-ned-funksjon. Den laster ned den gjevne git’en som spesifisert med kommandoen. Git’en inneheld all tidlegare historikk og kommandoen opprettar ein personleg local-git som starter å spore dine endringar.

Ein clone'et git kan verta oppdatert med kommandoen `fetch` og prosjektet henter då endringar som har blitt gjort av eigaren i det remote prosjektet.

`branch`[44]

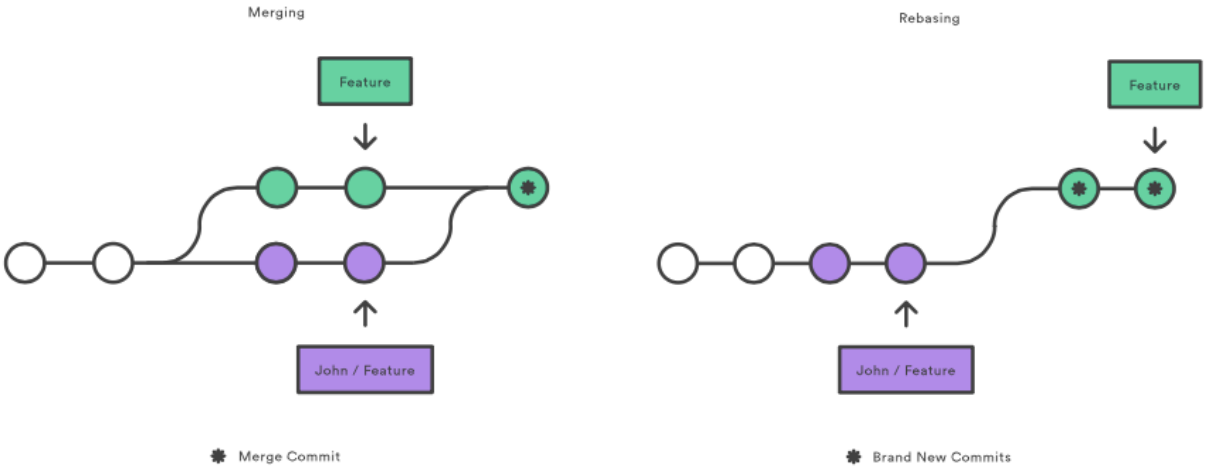
Når branch-kommandoen generere ei ny grein frå den noverande tilstanden til prosjektet, kan denne greina verta nytta til å prøve ut nye funksjonar utan å påverke det originale prosjektet.



Figur 50 Branching

`merge/rebase`[45]

Når ny funksjonalitet frå ei grein skal implementerast inn til master-greina kan du velje mellom enten ein merge eller ein rebase. Merge verkar på den måten at den beheld utviklingsprosessen i historikken i master-greina. Ved ein merge vert det generert ein merge-commit. På den andre måten vil ein rebase resultere i ei lineærisering av utviklingsprosessen og inkludere commit-historikken i master-greina.



Figur 51 Merge vs. Rebase

Det GitHub har gjort er å pare Git sin funksjonalitet med eit sosialt nettverk. Her vart det utvikla ein omgjevnad rundt det å dele og hjelpe andre med kodinga[46]. GitHub har indusert nye metodar på Git og presenterer alt i ei nettside. Sjølve flagship-metodane til GitHub er fork.

Fork er ein metode som kopierer det delte prosjektet og lagrar det på din GitHub side. Etter ein har fork'a eit prosjekt er ein fri til å endre i koden. Dersom ein føler at endringane ein har gjort vil bidra til prosjektet kan ein spør eigaren av prosjektet om å implementera dine endringar. Dette vert gjort via ein pull-request. Gjennom pull-request'en får eigaren av prosjektet opp endringane som er gjort og kan velja å merge desse inn i prosjekta. Personen som har bidrege med denne pull-request'en vil verta kreditert i prosjektet og på si personlege GitHub side. På denne måten har brukarar av GitHub høve til å opparbeide seg ein form for regime på prosjekter dei har bidrege på samt vist ferdighetane sine gjennom sine eigne prosjekter.

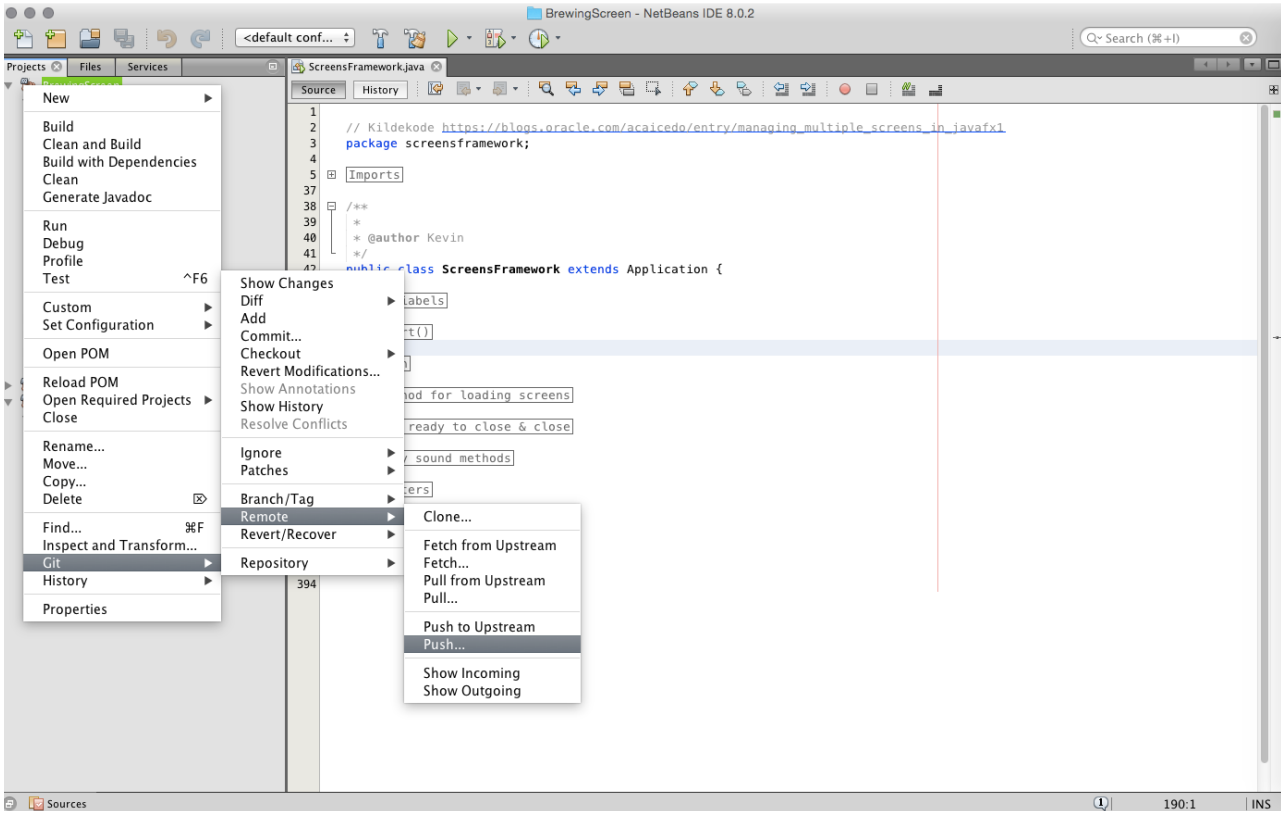
GitHub gir òg brukaren høve til å opne prosjektet for personar den trur vil bidra til prosjektet. Dette vert gjort ved å gjere personen til ein "Collaborators" og då vil personen kunne "pushe" si grein til master utan å sende ein pull-request.

Korleis Git vart nytta i prosjektet vår

Vi valde å nytte Git som versjonerings-system av di vi ikkje hadde vore borti nokre andre system før. Git er sterkt inkludert i Netbeans som var den IDE'en (Integrated Development Environment) vi nytta. Vi var eit lite team så vi fekk ikkje nytta alle funksjonane i GitHub, men tykkjer likevel at dette var eit viktig verkty til utviklinga vår. GitHub har bidrege med å halde oss oppdaterte på kvarandre sine kodar og begge har kunna jobba på same versjon. Dette sparte oss mykje arbeid når vi måtte gjere endringar den andre venta på. Vi kunne raskt oppdatere programmet, commit'e og push'e slik at den andre kunne pull'e og få tilgang til det den venta på.

Branchinga gjorde det enklare å brekke av ein del av programmet for å utføre tester og på denne måten kunne vi prøve nye ting uten å ta hensyn til å potensielt øydelegge hovudprogrammet.

Figur 52 på neste side viser korleis Git er integrert i NetBeans.



Figur 52 Git i NetBeans

3.3.3 Database

Kvifor database

Databaser er den vanlegaste måten for å lagre store mengder data og det finst svært få system som ikkje nyttar seg av dette. Vi ville nytte ein database til å lagre oppskrift-filene på ein organisert måte. Orginalt ville vi ha denne databasen på RPi, men sidan vi fekk litt lita tid laga vi RecipeMaker til bruk på separat datamaskin. Dette gjorde at vi måtte nytte ein remote-server som kunne hoste denne databasen. Vi valde mySQL som database av di vi hadde tidlegare kunnskap til denne. Vi ville lage to tabellar i databasen, ein med alle oppskriftene og ein med alle brukerene. Ved å nytte ein database kunne vi enkelt kopla oppskriftene saman med brukaren og berre gi brukaren tilgang til sine oppskrifter.

Oppretting og konfigurering av Databasen

Vi oppretta databasen på ein ekstern server i Nederland gjennom eit firma som heite DigitalOcean. DigitalOcean lar kunden konfigurere ein virtuell maskin som brukarane så får tilgang til gjennom SSH (meir om SSH under RPi Konfigurering). Vi konfigurerte vår “maskin” med 20GB SSD Disk 512MB RAM med Ubuntu 14.04 x64 som OS, vi velde òg å

ha serveren lokalisert i Nederland. På denne serveren vart databasen laga og vi laga to brukarar som skulle få tilgang til databasen. Vi gav brukaren “AutoBrygg” rettigheiter til å slette, oppdatere, opprette og motta rader i databasen.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON AutoBryggTest.* TO AutoBrygg@‘%’
IDENTIFIED BY ‘*****’ ;
```

AutoBrygg er brukaren vi nytter vidare i programma. Vi oppretta òg ein admin-brukar som har alle rettigheiter. Denne brukaren vart knytt til ei spesifikk IP-adresse for å auka tryggleiken. Etter at alle brukarane var etablert kunne vi gå i gang med å lage tabellane.
I Recipe-tabellen ville vi ha all info om oppskrifta i eigne kolloner til bruk i program, slik at vi ikkje trengde å laste inn heile oppskrifta før det var naudsynt.

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ID        | varchar(45) | NO   | PRI | NULL    |       |
| UserID    | int(11)     | NO   |     | NULL    |       |
| Type      | varchar(45) | NO   |     | NULL    |       |
| Name      | varchar(45) | NO   | MUL | NULL    |       |
| File      | blob        | NO   |     | NULL    |       |
| Created   | datetime    | NO   |     | NULL    |       |
| Modified  | datetime    | NO   |     | NULL    |       |
| BrewCount | int(11)     | YES  |     | NULL    |       |
+-----+--------+---+-----+---------+-------+-------+-----+
```

Vi ville ikkje at det skulle vere mogleg å ha to oppskrifter med same namn eller at brukarar skulle «bruke opp» namn og dette løyste vi med å gjere Name, i lag med UserID, unik.

User-tabellen vil vere kopla til Recipe-tabellen gjennom Recipe.UserID og Users.ID. Denne tabellen vil innehalda alt informasjon om brukaren. Tabellen vil verta nytta for å verifisera brukaren og sette UserID’en for å servere dei rette brukaroppskriftene.

```
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| ID       | int(11)     | NO   | PRI | NULL    | auto_increment |
| UserName | varchar(45) | NO   | UNI | NULL    |                |
| Name     | varchar(45) | NO   |     | NULL    |                |
| SirName  | varchar(45) | NO   |     | NULL    |                |
| Mail     | varchar(45) | NO   | UNI | NULL    |                |
| Password | varchar(45) | NO   |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

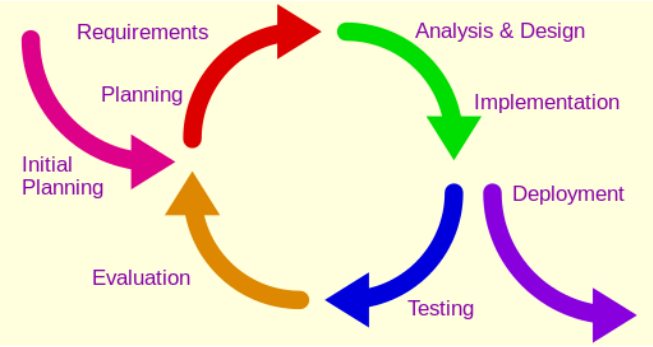
Forbetringar
Under neste revisjon av programmet ynskjer vi å skjule databasen frå nettet og heller gjere metodane synlege gjennom web-services.

3.3.4 Logikk

Planlegging
For å kunne planlegge logikken, og generelt eit slikt bryggeri, treng ein helst noko basiskunnskap i bryggemetoden. Ved tidlegare relevant prosjektarbeid deltok vi på eit tre dagars bryggekurs/seminar hos samarbeidspartnaren i prosjektet. Dette for å få ei grundig opplæring i brygging, samt at han då fekk forklart ein del om kva funksjonar han ynskjer i eit automatisert system. Etter kurset følte vi at vi hadde fått ei god innføring og vi kunne då starte planlegginga.

Under planlegginga var vi stadig i kontakt med samarbeidspartnaren. Kva krav han har kring funksjonane til systemet, feilmeldingar med tanke på utsjånad og handtering av desse, og mykje anna som dukka opp undervegs.

Vi starta med stykke opp bryggeprosessen i mindre delar. Dette for å gjere det enklare for oss å bygge opp klassar for små delar av prosessen for så å heller implementera desse i lag etter kvart. Det var òg semje i gruppa om at vi skulle følgje Iterasjonsmodellen[47] for kvart ledd. Ved å bruke denne modellen kunne vi jobbe med ein liten del av programmet i gangen, teste programmet og prøve ulike metodar for å løyse same problem.



Figur 53 Iterasjonsmodellen

Etter oppdelinga stod vi igjen med desse trinna:

1. Analog Input
    1. Temperature
    2. Level
2. Digital Output
    1. Pumps
    2. Valves
    3. Misc

3. Digital Input
    1. Valves
    2. Buttons
4. PID
5. SSR
6. Screens
    1. FXML
    2. Controllers
7. Main
    1. Fill water
    2. Mash
    3. Lautering
    4. Boil
    5. Cooling
    6. Drain
    7. Error

Det neste vi måtte løyse er korleis vi skulle få til kvar av desse klassene. Vi hadde allereie funnet utstyret som skulle vere i skapet, no måtte vi prøve å finne måtar å få desse til å snakke med RPi.

ADC Pi Plus og IO Pi Plus som skulle nyttast til I/Oar hadde ikkje Java-bibliotek, men hadde bibliotek til Python. Eit raskt søk avslørte at det ville vere mogleg å køyre eit Python-program i Java.

Framgangsmåte
For å løyse grunnlogikken bak kvar av desse ledda laga vi eit enkelt program for å teste ut ulike metodar. Når vi var nøgd med grunnfunksjonaliteten sette vi oss ned og drøfta element dette ledde måtte innehalde, korleis vi skulle hente ut informasjon frå klassen og korleis den skal behandle data. Når alt dette var på plass kunne vi setje igang raffinering av den gjevne klassen. Desse stega vart gjentatt med fleire av klassane til vi starta samansettinga.

Vi brukte ein vasskokar for å simulere delar av prosessen. På denne måten kunne vi gjenskape forholda i ein bryggeprosess, noko som gjorde det lettare å debugge programmet.

Analog Input (AI)
Denne mappa inneheld to klasser, temperatur og nivå, som førebehandlar dei analoge inngangane slik at logikken kan hente ut ferdigbehandla verdiar. Desse klassene er “runable" som vil sei at dei kan bli kjørt jamt i bakgrunnen. På denne måten vil klassane alltid innehalde oppdaterte verdiar. Klassane har ei feilsikring som fortel om målinga er truverdig. På denne måten vil det vere enkelt å overvake desse systema vidare i logikken.

Temperatur:
Denne klassen behandlar all logikk bak temperatur-innhenting. Her blir APIen lucidio.jar bruka. Dette er ein API som produsentane bak Lucid-modulen har laga[48]. Her er det lagt inn ytterlegare feilbehandling, dersom RPi ikkje får kontakt med USB modulen og om HERMS-coilen blir for varm vil det komme opp ein alarmskjerm. Klassen oppdaterer ein array med tre element gjennom ein fifo. Det er laga getters for å returnere enten array`en eller gjennomsnittet av desse verdiane.

Nivå:
Denne klassen fungere på utsida veldig likt temperatur klassen, men innsida er bygd opp på ein heilt anna måte. Av di det ikkje eksisterte noko bibliotek til Java måtte vi nytte oss av Python til å få kontakt med ADCen. Når biblioteket blir etablert vil vi implementere dette for å få eit reint Java-programmeringsmiljø. Python programmet blir køyrt via Runtime klassen sin exec metode og deretter hentar vi input streamen til prosessen og les inn verdiane til Java. Dersom klassen ikkje har fått ein truverdig verdi frå ein sensor fem ganger på rad vil det resultere i ein alarmskjerm.

Digital Output (DO)
Denne mappa har fire klassar og er atterhald til styring av eit IO Pi Plus kort, som vert nytta til utgangar. Mappa inneheld klassene ErrorShutDown, Misc, MotorPumpController og ValveController. Klassane nytter Python-program til å setje utgangane.

ValveController:
ValveController-klassen er den største klassen i denne mappa. Den inneheld ein metode for kvar ventilstilling som er i prosessen. Metodane tar ein boolean som parameter for å avgjere om den skal opne eller lukke. Metoden returnerer true om ventilen får stadfesta rett stilling eller om override-brytaren er inne, elles returnerer den false og setter inn feilkoden som blir lasta seinare i logikken.

MotorPumpController:
Denne klassen er lik på ValveController-klassen. Den har ein metode for kvar pumpe, men den har ikkje noko stadfesting på om pumpa køyrer så den returnerer ikkje noko.

Misc:
Denne klassen er foreløpig til å setje override-lyset.

ErrorShutDown:
Klassen inneheld ein metode som stenger alle “farlige” ventilar og motorar. Metoden blir kalla på ved feil i prosessen og set så eit feil-indikator lys. Dette lyset er kun til test. Det er òg ein metode for å resette feil-indikator lyset.

Digital Input (DI)
I denne mappa er det kun ein klasse og denne tar seg av alle digitale inngangar.

Control:
Klassen implementerer Runable som gjer at prosessen kan bli køyrt i bakgrunnen for å oppdatere sine variablar. Her blir det brukt Python på same måte som i AI.LevelM-klassen. Control-klassen grupperer verdiane slik at vi kan kalle på ein metode som sjekkar om alle ventilane i det gjevne trinnet er i rett tilstand i staden for å skjekke kvar enkelt ventil.

PID
PID mappa inneheld to klasser og blir brukt under regulering av effekt til varmeelementa. Mappa har ein klasse som reknar ut pådrag og ein enum som set PID-parameter.

PID:
PID-klassen sin calc-metode tar ein double som parameter og returnerer ein double. Metoden køyrer parameteren gjennom ein PID-algoritme og returnerer svaret. Metoden har nokre avgrensingar for å unngå “windup” dsom blir gjort ved å avgrense den akkumulerte I-feilen.

PIDParameters:
Denne klassen er ein enum som inneheld PID-parameter som blir brukt av PID-klassen. Vi valde å bruke ein enum her for å lett kunne endre verdiane. Det vil òg vere ein spesifikk variabel for kvart steg i prosessen. Dette er fordi det er forskjellige krav for kvart steg der nokre krev rask oppvarming medan andre krev at det ikkje skal vere fare for oversvinging.

SSR
Dette er klassen som styrer sjølve pådraget, som blir gjort ved å pulsere RPi-utgangar av og på. Her blir biblioteket pi4j nytta til å få tilgang i GPIOane[49]. GPIOane måtte vere kopla direkte inn på RPien for å kunne møte tidskravet vi hadde på pulseringa. Vi har sett opp denne klassen slik at det må bli laga eit objekt med GpioPinDigitalOutput[50] og int som parameter. GpioPinDigitalOutput fortel klassen kva utgang som skal bli styrt og int-verdien som blir sett er startpådraget. Klassen blir køyrt i ein separat tråd som styrer pulsane ut til varmeelementet.

Screensframework
Denne samlinga inneheld ei rekke klassar som er brukt til å illustrere logikken. I denne samlinga er det to interface som blir nytta for å passe på at klassane har dei metodane som er obligatoriske.

ScreensFramework:
ScreensFramework inneheld main-metoden. Denne klassen lagar og startar trådane til informasjonsinnhenting og opnar kommunikasjonen med databasen. Denne koplinga blir så gjort tilgjengeleg til andre klassar gjennom ein “getter”. Her er det òg ei rekke metodar

for å laste ulike skjermar. Denne klassen startar òg opp lydklassane og gir spelemoglegheitene tilgjengeleg for andre klassar.

ScreenXXController:
Dette er ei rekke med klasser som er ganske like. Primært brukt til å styre skjermane. Klassane har vi prøvd å holde mest mogleg separat frå logikken i henhald til MVC-modellen[51]. MVC-modellen tilseier at kontrolleren ikkje skal innehalde logikk som behandlar modell-data. Kontrolleren hentar kun ut data frå modellen og kan klarerer ei feilmelding.

Interface'ane:
Desse to interface-klassane (som nemnt tidlegare) er for å passe på at alle klassane følgjer ein viss struktur. På denne måten blir programmet meir forutsigbart og gjer det lettare å halde orden på kjente metodar i klassane. I framtida ynskjer vi å implementere fleire metodar inn i interface'en, men det er ikkje støtte for private metodar enda, noko som gjorde at vi valde å ikkje gjere dette med fleire. Støtte for dette kjem i Java 9[52].

InfoRecipeScreenController:
Dette er ein relativt ny klasse. Her har vi brukt mange element frå RecipeMaker for å lage tabellar med ingrediensar som skal bli tilsett under prosessen. Tabellane er kategorisert i grupper som gjer det enkelt å kontrollere om ein har dei rette ingrediensane klare. Denne skjermen er meint som eit valfritt trinn til bruk dersom ein har gløymt å gjere klart alle ingrediensane eller om ein vil dobbeltsjekke noko før brygginga startar.

Logic
I Logic mappa har vi plassert hovudlogikken. Klassane er delt opp i trinna som er i prosessen. Det er òg nokre hjelpeklassar her som blir brukt i mange av klassane. Klassene blir køyrt som trådar og blir kun instansiert ved bohov.

Controller00Misc:
Dette er ein modell som vert nytta av mange skjermar. Klassen passar på verdiar som er viktige å passe på under heile prosessen. Denne modellen blir instansiert og køyrt frå starten av oppstart og er med vidare heilt til programmet avsluttes.
Controller01FillWater:
Denne kontrolleren vert nytta til å starte prosessen. Den fyller fyrst 1/3 av mesk-inn vatnet med varmt vant og resten med kaldt vant. Det blir så sett igang sirkulering og oppvarming slik vatnet har rett temperatur når maltet skal tilsetjast. Når temperaturen er nådd blir sirkuleringa slått av slik at ein kan røre inn malt. Klassen inneheld òg feilbehandling for ventil-stillingar.

Controller02Mash:
Denne klassen blir brukt under meskingen. Den kontrollerer pådraget til varmeelementet, passar på tida og set eventuelt ny mesketemperatur. Når klassen blir instansiert tar den ei

ArrayList med CommonMashingClass som parameter. Denne lista blir nytta til å hente ut temperaturar og mesketid. Når klassen har fått naudsynt informasjon startar oppvarminga til den gjevne temperaturen og når temperaturen er nådd startar nedtellinga. Nedtellinga av mesketida vil bli sett på pause dersom temperaturen fell meir enn to gangar toleransen. Ved feilmelding vil meskeprosessen fortsatt vere igang sjølv om sirkulering og oppvarming stoppar. Det er av den grunn nedtellinga fortsetter ved feil. Den blir kun sett på pause dersom temperaturen fell under terskelen. Når nedtellinga er ferdig skjekker klassen om det er fleire trinn i lista og om så startar prosessen på ny.

Controller03Lautering:
Lautering-kontrolleren tar seg av logikken bak skylje-prosessen. Parametera som denne klassen treng er temperaturen på skyljevatnet og nivået som skal vere i kokekaret. Det fyrste som blir gjort er å starte tappinga ut av meskekaret og deretter opne for innløpet av skyljevatn. Når eit grensenivå er nådd stenges innløpet for skyljevatnet medan uttappinga fortsetter. Tilslutt, når nivået er nådd i kokekaret, signaliserer klassen at neste skjerm kan bli lasta. Klassen inneheld òg feilbehandling for ventil-stillingar.

Controller04Boil:
Instansieringa av denne klassen krev to lister. Parameteren inneheld informasjonen om alt som skal bli tilsett under kokinga. Det fyrste denne klassen gjer er å finne fram dei fyrste tilsetjingane som skal bli tilsett. Desse blir lagt inn i ein egen tabell som skjerm-kontrolleren kan hente for å vise fram. Når kokekaret har nådd ei rullande koking startar nedtellinga og når det er tid for å tilsetje ingrediensar speler den av ein lyd. Listene med ingrediensar blir ikkje oppdatert før det har gått eit minutt. Klassen opererer òg på tid så her er det viktig å pause klassen i den tilstanden den er i, om noko skulle oppstå. Femten minutt før kokinga er ferdig blir det sett igang noko kalla wirlpooling.

Controller05Cooling:
Klassen finn temperaturen vørteren skal kjølast ned til ved å ta alle øvre gjæringstemperaturar og vel den lågaste som set-punktet til programmet. Vørteren blir kjølt ned til set-punktet er nådd. Etter dette stopper den sirkulasjonen og venter ei tid (“settling time”) før den gir beskjed om at den er ferdig.

Controller06Draining:
Denne klassen passer på at bryggar ikkje kan gå vidare før eit minimumsnivå er nådd.

Stats:
Dette er ein enum for å gjere programmet meir leseleg og blir nytta til å byte ut orda true/false til ventilane og pumpene, med OPEN/CLOSE og ON/OFF.

CountDownTimer:
Denne klassen blir nytta i mange andre klassar for å telje ned. Den blir med anna mykje brukt i ventilkontroller-klassen for å kontrollere at ventilane held tidskrava.

DateAndTime:
Denne klassen blir nytta av mange klasser for sette dato og tid på skjerm. Denne klassen er Runnable og blir oppdatert med gjevne mellomrom.

Forbetringar
Det er fortsatt mykje som kunne vorte forbetra i programmet. Vonleg vil det snart kome eit Java-bibliotek til utvidingsmodulane. Når desse biblioteka vert tilgjengelege vil gruppa implementere desse og fjerne bruk av Python.

Ein annan ting vi skulle gjort er å lage nokre “listeners" på kritiske delar i programmet, til dømes til naudstopp eller andre grenseverdiar.

Vi ville òg gjort litt om på behandlinga av tilsetjingar under kokeprosessen. Her kunne vi gjort det likt som i meske prosessen. På denne måten kunne vi hatt svak tekst på tilsetjingar som har vorte tilsett og tydeleg tekst på tilsetjingar som skal tilsetjast. Problemet er at det kan bli mange tilsetjingar i ei oppskrift, meir enn vi klarer å vise på ein skjerm, difor valde vi å kun vise dei aktuelle tilsetjingane. Forbetringa vil bli å lage denne delen av programmet meir dynamisk slik at dei nye elementa skyv ut dei gamle og ein form for variabel som fortel view controller’en kva tilsetjing som skal vere i svak/sterk tekst.

Ein liten funksjon som òg kan bli lagt til er å nytte ei lydvarsling litt før ingrediensar skal tilsetjast slik at bryggar kan gjere desse klar.

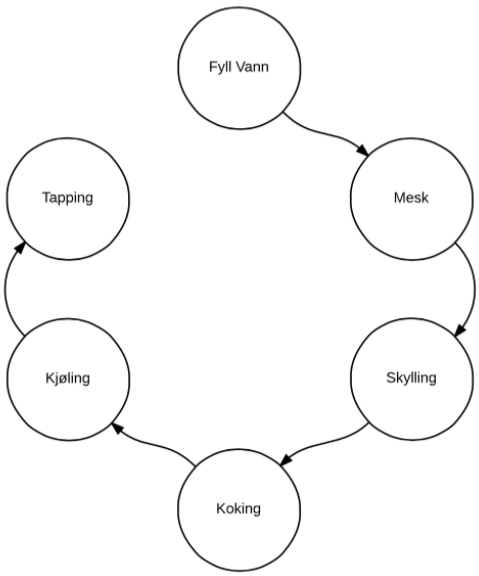
3.3.5 BrewingScreen

Intro

BrewingScreen er ein applikasjon som køyrast på bryggesystemet. Ved oppstart må brukar logge seg inn med brukarnamn og passord for å få tilgang til sine oppskrifter. Oppskriftene vert henta frå databasen der dei er lagra og vist på bryggeskjermen. Brukaren vel ei oppskrift og bryggesystemet vil køyre igang bryggeprosessen, steg for steg.

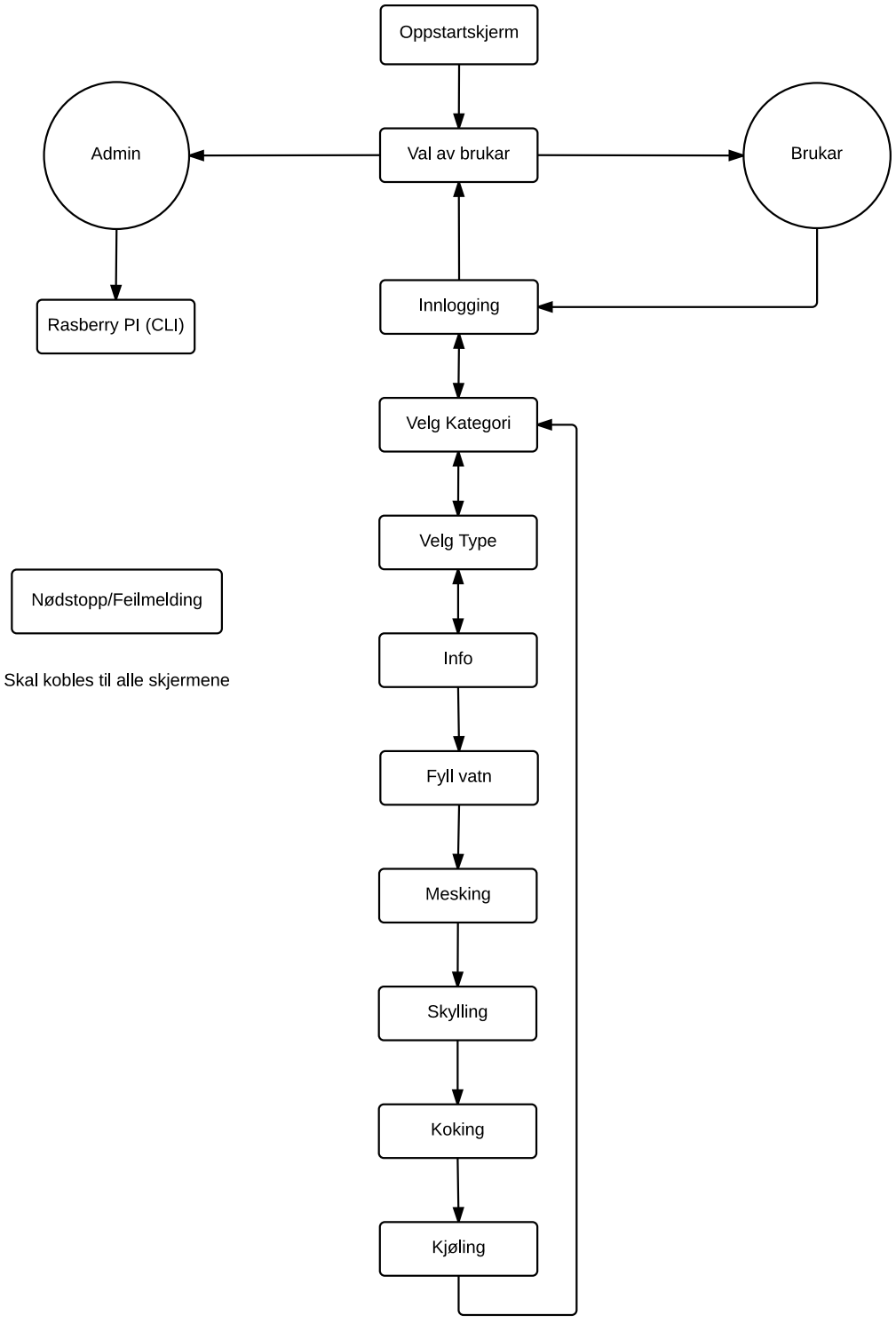
Planlegging av programmet

Dei fleste bryggeprosessar nyttar den same framgangsmåten for å ferdigstille eit brygg. Figur 54 viser dei grunnleggande stega ein må igjennom. Gruppa ynskja å utvikle eit enkelt og brukarvenleg program slik at brukar kunne kjenne att dei grunnleggande stega. Det fyrste vi gjorde var å sette oss inn i prosessteikningane. Vi teikna deretter opp kvart steg og kva det inneheldt av gryter, ventilar, pumper og varmeelement. Dette vil gjere det enklare for gruppa når vi skal byrje å prosjektere innhaldet i skjermane og programmeringa av logikken til programmet.

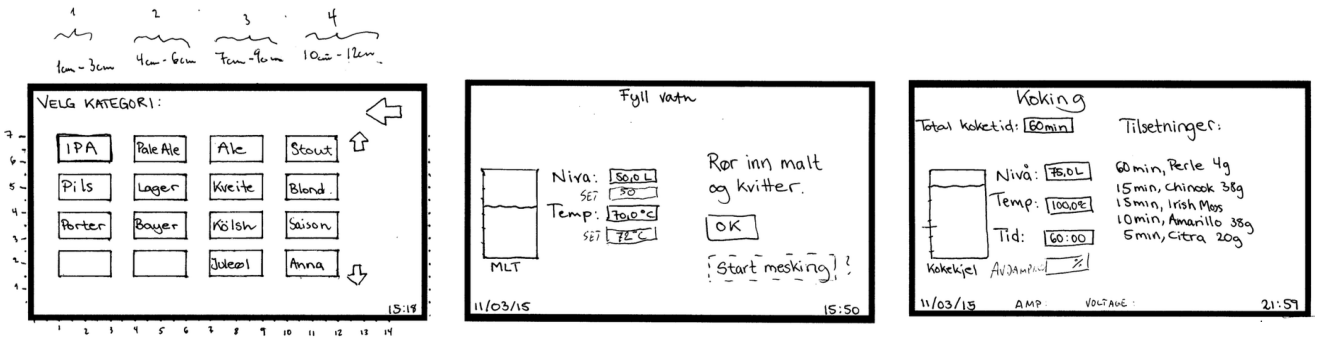


Figur 54 Steg i bryggeprosessen

Vi laga eit flytskjema over skjermane som skulle visast på touch-skjermen, vist i figur 55. Flytskjemaet gjorde arbeidet vidare ryddig og oversiktleg sidan vi viste korleis programmet skulle byggast opp. Det neste gruppa gjorde var å fylle ut alt innhaldet i skjermane. Vi såg på teikningane over dei ulike stega i prosessen for å hente ut den nødvendige informasjonen skjermane skulle trenge med tanke på temperatur, nivå og tid. Nokre dømer på skjermar er vist med figur 56. Skjermane vart presentert for samarbeidspartnar 14.03.2015 og protokoll frå møtet finn du i vedlegg 6, protokoll 4. Samarbeidspartnar ynskja litt meir informasjon på skjermane, noko gruppa ordna og fekk godkjent 20.03.2015. Protokoll frå møtet finn du i vedlegg 6, protokoll 5.



Figur 55 Fyrste utkast av flytskjema

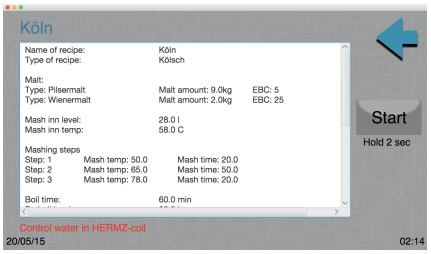


Figur 56 Skissering av skjermbilete

Når teikningane av skjermane var godkjende starta gruppa å programmere desse. Skjermane vart laga i «Scene builder» som er eit «drag and dropp»-program[53]. Dette er eit verkty der ein har høve til å lage skjermar utan koding. Derimot må ein programmera funksjonaliteten og dette vart gjort i java.

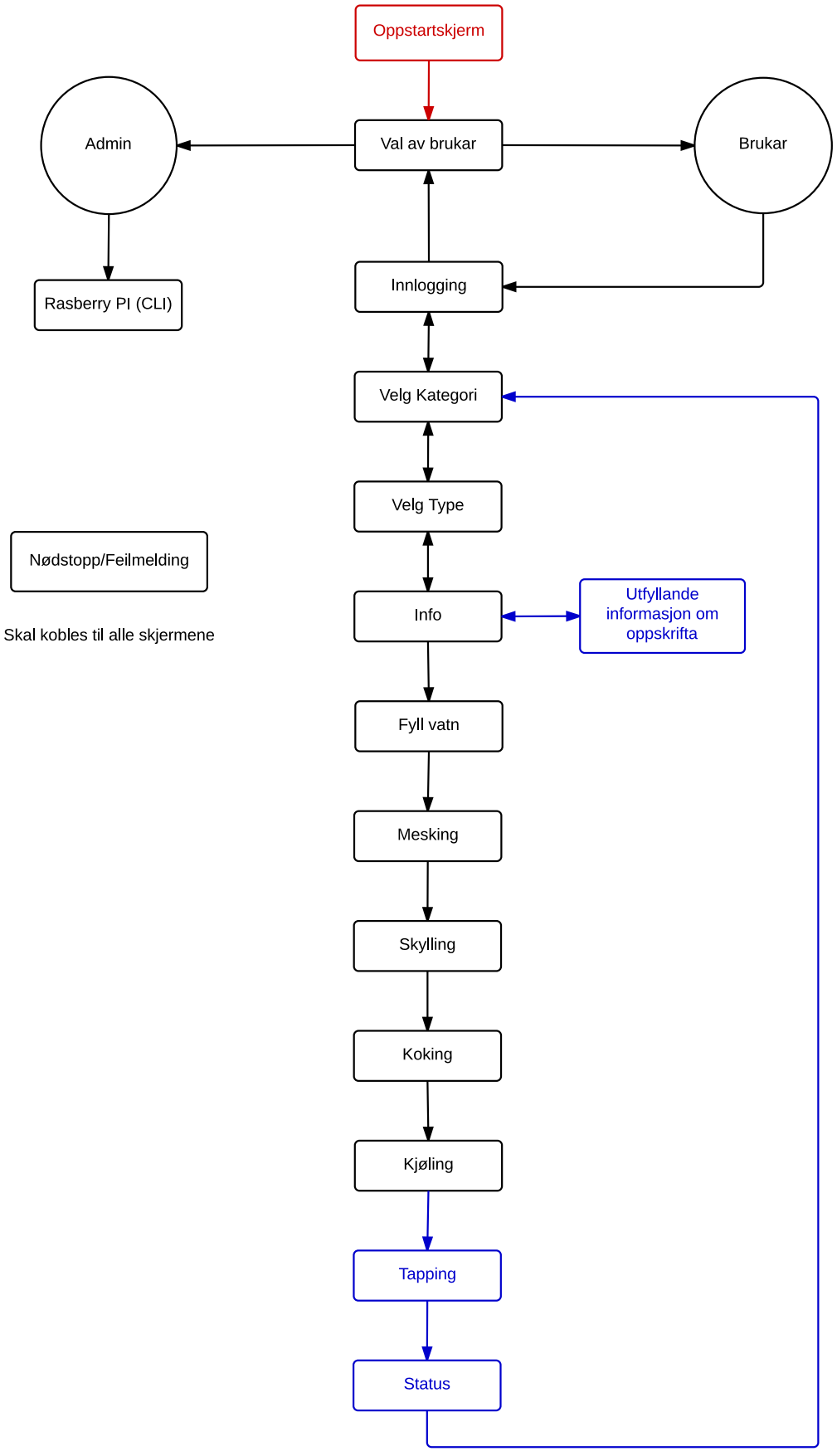
Endringar

Gruppa såg ikkje noko hensikt i å ha ein oppstartskjerm og valde difor å fjerne denne. Figur 58 viser korleis funksjonaliteten ser ut etter gruppa var ferdig med programmet. Under testing valde gruppa å gjere nokre endringar. Info-skjermen skulle presentere oppskrift i eit tekstområde, vist i figur 57. Vi følte at dette var lite oversiktleg og valde difor å endre denne visninga. Gruppa løyste dette med å berre vise den viktigaste informasjonen på info-skjermen og lagde ein ny skjerm for den resterande informasjonen.



Figur 57 Infoskjerm

Gruppa valde òg å gjere nokre endringar i slutten av programmet. Vi velde å ta vekk tappefunksjonen i kjøle-skjermen og heller gjere den til ein ekstern del, ein eigen skjerm. Samtidig la vi til ein avsluttande skjerm med informasjon om brygget. Brukar får òg to valmoglegheitar der han kan velje å anten starte eit nytt brygg eller avslutte programmet.



Figur 58 Endeleg flyskjema

Gjennomgang av skjermane

Når brukar slår anlegget over på AUTO vil skjermen skru seg på. På det fyrste skjermbilete må brukar velja mellom knappane «BRUKAR» og «ADMIN». Dersom ein vel «ADMIN» vil programmet avsluttast og ein vil hamne i kommandovindauget til RPI (CLI)[54]. Dersom brukar vel den andre knappen vil ein ny skjerm kome fram og brukar må taste inn brukarnamn og passord for å logge seg inn.

Programmet vil så henta alle bryggekategoriane som er knytt til brukarnamnet og presentere dei. Når brukar vel ein av kategoriane vil ein ny skjerm presentere alle oppskriftene som er knytt til kategorien. Brukar vel ei oppskrift og innhaldet i oppskrifta vert presentert. Brukaren kan no starte bryggeprosessen.

Når bryggeprosessen startar vil ein skjerm for kvar del i bryggeprosessen bli presentert. Skjermane inneheld den nødvendige informasjonen ein måtte trenge, t.d temperatur, nivå og tid. På nokre skjermar vil nivået bli simulert og viktige tider varsla med lyd.

Når kjøleprosessen er ferdig vil skjermen med tapping bli presentert. Brukar har då høve for å tappe over på gjæringskar. Dersom det skulle vere nødvendig har brukar høve til å stoppe tappinga og starte den igjen. Når dette er gjort vil den siste skjermen visast for brukaren. Den inneheld nykelinformasjon om brygget. Ein kan så velje å anten gå tilbake til bryggekategoriane eller avslutte programmet.

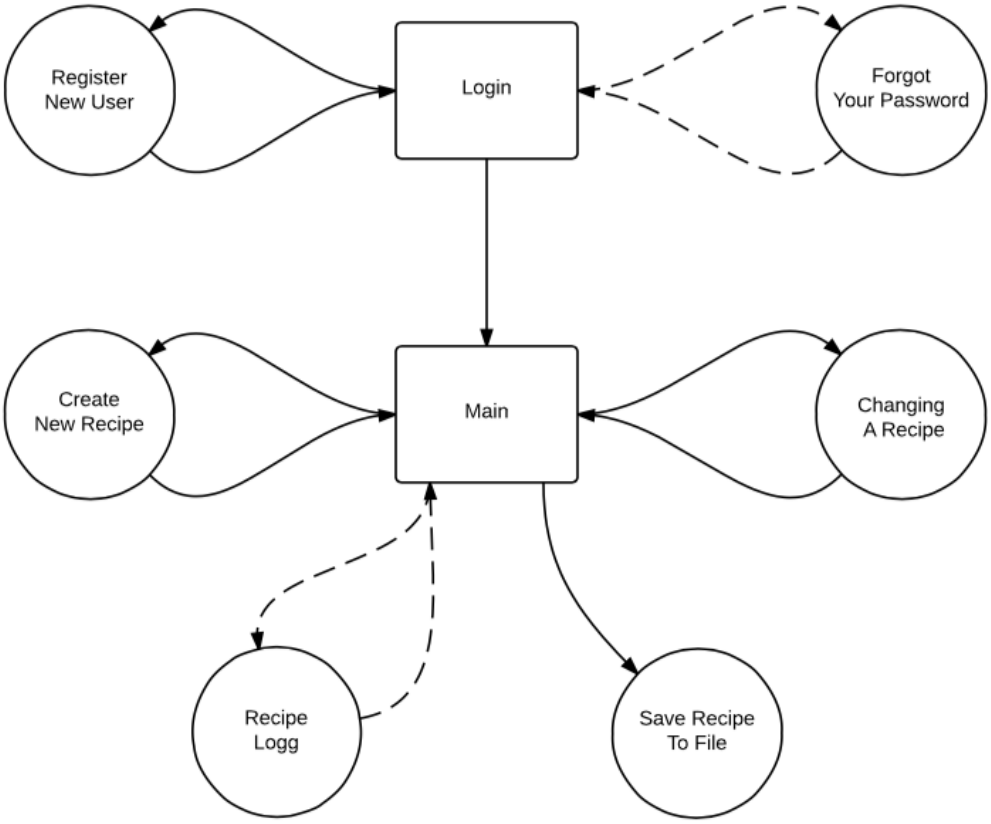
Dersom det skulle skje noko uynskt eller at naudstoppen vart trykka inn, vil ein skjerm med informasjon dukke opp.

3.3.6 RecipeMaker

Intro
RecipeMaker er ein applikasjon der kvar enkelt brukar har moglegheit til å lage, endre og organisere sine oppskrifter. Denne applikasjonen kan køyrast på brukaren sin private PC, noko som gjer at brukaren av bryggesystemet har høve til å planlegge og bestille inn det dei måtte trenge på førehand. Oppskrifter som vert oppretta i RecipeMaker vert lagra på ein database. Både RecipeMaker og BrewingScreen er knytt til denne databasen. Når brukar logger seg inn på BrewingScreen vil brukar sine oppskrifter bli henta og vist på skjermen.

Planlegging av programmet
Då gruppa skulle starte på utviklinga av RecipeMaker vart det sett av mykje tid til sjølve planleggingsprosessen. Det er fleire grunnar til at gruppa valde å gå denne vegen. Hovudgrunnen er prosjektet sine tidsrammer, noko som gjer gruppa sitt samarbeid ekstra viktig. Heile gruppa vil under prosjektperioden vite kva som er planen for programmet sin funksjonalitet. Det vil òg skape eit betre samarbeid og ikkje minst vil framtidige feil som kan oppstå reduserast.

Gruppa starta med å skrive ned alle funksjonane vi ville at programmet skulle innehalda. Deretter laga vi eit flytskjema, vist i figur 59. Flytskjemaet har heile tida vert ein sentral del i utviklinga av programmet.



Figur 59 Flytskjema

Login

For å nytte programmet må ein opprette ein brukarkonto. Dette vert gjort i sjølve programmet der ein fyller ut eit skjema. Det er ikkje mogleg å registrera to brukarar med same brukernamn eller epost. Kvar konto er unik og ein kan ikkje sjå eller bruke andre sine oppskrifter. Dette vert gjort for at brukar ikkje skal kunne slette eller endre andre sine oppskrifter samstundes som at det vil gjere programmet meir personleg.

Utfyllingsskjemaet skal innehalda:

- Namn
- Etternamn
- Brukarnamn (unikt)
- Passord
- Bekreft passord
- Epost (unikt)

Main
Denne delen av programmet skal bestå av fleire ulike delar. Brukaren skal ha tilgang til ein brukarmanual som skal innehalde all naudsynt informasjon ein treng for å nytta programmet. Oppskriftene skal presenterast på ein oversiktleg måte og vere delt opp i ulike kategoriar. Det skal òg vere mogleg å endre og lage nye oppskrifter, samt lagre oppskriftene til en fil.

Vidare utvikling
Under planleggingsprosessen kom gruppa opp med mange gode idear på kva programmet måtte innehalde. Gruppa vedtok til slutt å avgrense programmet noko, men ynskja ikkje å fjerne alle ideane. Vi sette difor opp to ting vi ynskja å vidareutvikla seinere dersom tida ikkje skulle rekke til. Den fyrste tingen var å resette passordet ved at brukar skriv inn epostadressa si. Brukar skulle då få tilsendt eit mellombels passord som brukaren kunne bruke og lage seg eit nytt. Den andre ideen var å lage ein bryggelogg. Gruppa tykkjer denne ideen var veldig viktig, men anlegget må testast ut nokre gangar før ein veit korleis ein kan bygge denne loggen opp.

Programmet sine klassar
Package no.autobrygg.recipemaker inneheld åtte ulike klassar. Desse klassane administrerer og set skjermane sitt innhald. Når ein skal jobbe med denne typen data er det viktig å tenke på programmet si yting. Dersom programmet skulle utvikle seg til å bli stort med mange skjermar vil dette påverke programmet si yting og brukaren si brukaroppleving. «Angela´s Blog»[55] tar opp akkurat denne problemstillinga. Ho skildrar problematikken rundt denne typen program. Vi valde å bruke hennar løysing for å administrera skjermane i våre program.

Angela si løysing bygger på strukturen StackPane. Ho tar for seg ulempene med denne strukturen, men beskriver òg korleis den kan brukast på ein litt annleis måte.

ScreensFramework
Denne klassen inneheld naudsynt informasjon for å hente dei ulike fxml-filene som er kopla til dei ulike skjermane. Fxml-filene inneheld skjermane sitt innhald i tekstform. Ved oppstart vil denne klassen opne forbindinga med databasen og vise den fyrste skjermen.

ScreensController
I dette prosjektet valde vi å gå for ein StackPane-struktur. Dette bygger på at vi legg skjermane oppå kvarandre. Vi ynskja ikkje å laste opp alle skjermane på ein gang då det ville ha påverka ytinga til programmet betrakteleg. «Angela's Blog» løyste dette med å lage ein liknande struktur der berre ein skjerm vert lasta opp og vist om gangen. Denne klassen inneheld naudsynte metodar for å behandle skjermane på ulike måtar.

ControlledScreen
Dette er ein interface-klasse som vert inkludert i kvar skjermkontroller.

Screen01Controller
Denne klassen sjekker om utfyllingsfelta er fylt ut. Den sender informasjonen vidare til databasen der den vert behandla. Databasen gjer om passordet som brukaren har tasta inn til ein md5Hash tekst. Databasen hentar passordet som er kopla til brukarnamnet og samanliknar dei. Databasen gir så tilbakemelding om brukaren eksisterar eller ikkje. Den sjekker òg om brukar ynskjer å abonnere på AutoBrygg sine oppskrifter og sender det så vidare til databasen som registrerer det og gir brukar tilgang om det skulle vere ynskjeleg.

Screen02Controller
Denne klassen gjer to ting når den vert lasta inn:

1) Den hentar brukarmanualen frå klassen RecipeMakerUserManual og presenterer den i ulike kategoriar. Brukermanualen inneheld informasjon om programmet sin funksjonalitet og brukarveiledning.
2) Den hentar inn oppskriftene som er knytt til brukaren sitt id-nummer. Om brukar har abonnert på AutoBrygg sine oppskrifter vil desse òg bli henta. Dette blir henta frå databasen og presentert i ulike kategorier.

Klassen gir òg høve for å lagre oppskrifter på ei excel-fil.

Screen03Controller
Denne klassen kan lagre oppskrifter til databasen. Dersom det skal lagast ei ny oppskrift vil alle utfyllingsfelt vere blanke. Om ein ynskjer å endre på ei oppskrift vil denne klassen hente oppskrifta frå databasen og fylle ut utfyllingsfelta med den informasjonen som oppskrifta inneheld.

Screen04Controller
Denne klassen sjekker om alle utfyllingsfelta er fylt ut. Den sender informasjonen vidare til databasen der informasjonen vert behandla/sjekka. Databasen gir tilbakemelding om registreringa var vellukka.

Screen05Controller
Denne klassen er sett opp for vidare utvikling.

Package UserManual inneheld klassen RecipeMakerUserManual.

RecipeMakerUserManual
Denne klassen inneheld brukarmanualen til RecipeMaker i tekstform. Tanken bak denne klassen var at den skulle vere oversiktleg og lett å oppdatere ved vidare utvikling.

Package Validation inneheld tre ulike klassar. Det dei har til felles er at dei sjekkar og gir tilbakemelding.

CheckForNumberTextFieldValidation
Denne klassen har ein lyttar som vert tildelt eit utfyllingsfelt. Den sjekkar om utfyllingsfeltet og innskrivingsfeltet inneheld nummer og ikkje tekst.

EmptyTextFieldValidation
Denne klassen sjekkar om innskrivingsfeltet er tomt.

GValidation
Programmet BrewingScreen nyttar ein del av informasjonen som vert skrive inn i RecipeMaker. Det er difor viktig at nokre av desse verdiane ikkje er for små eller for store. SG (Specific gravity) er ein av desse verdiane. SG-verdien vert nytta til å rekne ut volumet i kokekjela før ein startar koking. Det er akkurat det denne klassen gjer. Den sjekkar om brukaren har skrive inn verdiar som er innanfor det som er sett som minimum og maksimum. Desse verdiane vert sett i denne klassen og kan bli henta av resten av programmet.

Package Fader inneheld klassen CreateFader.

CreateFader
Denne klassen vert nytta når ein ynskjer at teksten som viser skal falme vekk.

Package WriteRecipeExcelFile inneheld klassen WriteExcelFile.

WriteExcelFile
Denne klassen lager eit nytt Excelark. Den har òg ulike metodar for å fylle inn tekst. Den lukker så forbindinga og lagrar fila.

3.3.7 Common-klassane

Sidan vi hadde to program som skulle fordele data måtte vi ha nokre felles klassar som var like i begge programma. Ved å nytta common-klassar kunne vi lagre tilstanden til klassen til ei fil og deretter lese den frå fila i det andre programmet.

Common classes

Desse klassane vert nytta for å lage og lagre oppskriftene slik at begge programma kan hente ut informasjon. Klassane har “xml-annotation” noko som gjer at vi kan skrive/lese desse klassane til/frå ei .xlm fil. Klassene har òg ein toString-metode for rask og enkel utskrift.

CommonAdditivesClass og CommonHopClass
Desse klassane inneheld verdiane til tilsetjingar som vert nytta under kokeprosessen. Klassane vert nytta under kokeprosessen for å informera bryggaren om kva som skal tilsetjast og kva tid. Klassane har ein compareTo-metode som gjer at ei liste av objekt av klassane kan bli sortert etter koketida.

CommonMaltClass
Denne klassen er laga for lagre informasjon om kva malttypar ein skal nytte. Desse verdiane vert ikkje direkte nytta av bryggesystemet, men er tilgjengeleg for brukar å sjå før bryggeprosessen startar.

CommonYeastClass
Klassen inneheld informasjon til kva type gjær som skal nyttast etter bryggeprosessen og kva temperaturar gjæren er kompatibel med. Gjærtemperaturane vert nytta vidare i det automatiserte bryggesystemet for å bestemme kjøletemperaturen.

CommonMashClass
Denne klassen inneheld informasjon om meskinga. Her må vi å vite tid, temperatur og steg. Klassen har ein compareTo-metode som gjer at ei liste av objekt av denne klassen vert sortert etter meskesteg-rekkefølga.

Common
Dette er hovudklassen og den inneheld alt informasjonen om oppskrifta. Her er alle namna, temperatur og nivåverdiane lagra. Denne klassen inneheld òg ei liste av alle dei andre Common-klassane.

SQL
Denne klassen inneheld alle metodar for å kople, hente ut, redigere og slette innhald i databasen. Den inneheld òg ein metode for å kontrollere brukarnamn med passord samt kryptering av passord.

ClassForRecipeMaker
Denne klassen vert nytta av RecipeMaker for å representera data i databasen. Denne klassen vart generert frå databasen i Java.

XML

XML-klassen vert nytta til å lese, skrive og slette .xml filer. Skrivemetoden tar inn Common Objekt som parameter og skriv ut ei fil i programmet sitt direktorat.
Etter skrivemetoden vert nytta og fila er lasta opp til databasen kallar vi på deleteFile-metoden for å fjerne denne fila frå datamaskinen. Dette gjer vi fordi vi vil at andre brukarar skal kunne bruke den same dataen til å lage oppskrifter utan å kunne gå inn å lese .xml fila til en annan brukar. LoadDataFromInputStream-metoden les .xml fila som ein inputstream frå databasen og retunerar denne som eit objekt av common-klassen.

MiscUtill

padLeft
Denne metoden vert nytta for å lage tabulatur til toString-metodene i common-klassane.

generateUniqueID
Denne metoden vert nytta til å generere ein unik id til bruk som ID til Recipe-tabellen i databasen.

4.0 Måloppnåing

Hovudmålet til oppgåva var å planlegge og til dels prosjektere eit automatisk ølbryggesystem, med vekt på instrumentering, elektronikk og programmering. Vi føler at vi har nådd dette målet med god margin. Vi planla òg å sjå på kostnadane ved å bygge eit slikt system og korleis ein kan gjere det til eit kommersielt produkt. Dette punktet slo vi frå oss då det vart vanskeleg å avgjere ein total pris på anlegget i løpet av prosjektperioden samstundes som at det vart knapt med tid.

Vi hadde ved tidlegare prosjektarbeid allereie starta planlegginga av oppgåva. Denne planlegginga gav oss ein god grobotn når vi skulle starte på hovudoppgåva. Vi lærte oss grunnleggande bryggeprinsipp og undersøkte ein del på kva utstyr me kunne nytte til styring og elles i eit slikt systemet. Noko av det vi jobba med har vi kunna ta med oss vidare, men vi vil seie at vi likevel starta på “scratch” når vi starta på hovudoppgåva.

Saman med samarbeidspartnar planla vi korleis systemet skulle sjå ut og korleis det skulle virke. Ut frå dette har vi laga fullstendige prosessteikningar, koplingsteikningar og ei uttømande komponentliste. Dette gjorde vi tidleg i prosjektperioden, men av di det stadig har kome endringar (av ulike årsaker) har vi tilslutt brukt ein del meir tid på dette enn planlagt samt at dei ikkje vart klar til planlagt tid. Dette medførte at vi blei noko seine når vi skulle bestille inn utstyr til skåpbygginga.

Som eit delmål i prosjektet skulle vi bygge tavla/styreskåpet til systemet. Av di vi vart seine med ferdigstillinga av komponentlista og teikningane valde gruppa å ikkje stresse med dette og heller lage ein prototype med det utstyret vi hadde, noko samarbeidspartnar syns var ein god idé. Denne avgjersla er vi i grunn veldig nøgd med. Ved å lage prototypen fekk vi i større grad høve til å teste ut utstyr og program/logikk samstundes som vi får vist prinsippet (veldig forenkla) bak systemet på framføringa.

Programvaredelen, programmering av logikk, grensesnitt (UI) og database, har vi jobba veldig mykje med og utgjer nesten 40% av alt prosjektarbeidet.

Vi hadde som mål å lage ein bryggelogikk der brukar kan styre brygginga ved hjelp av ein toutch-skjerm. Programmet vi har laga er utforma som ein applikasjon kalla BrewingScreen, der vi la stor vekt på brukarvenleiken.

Vi planla at den delen av programmet der oppskriftene skulle skrivast inn skulle vere ein veldig enkel funksjon. Av di det vart litt ventetid på bygging av skåp valde vi å bruke meir tid på denne delen av programmet og heller lage ein fullstendig applikasjon. Med dette aukar vi brukarvenleiken for denne delen av programmet. Vi kalla applikasjonen for

RecipeMaker. Til denne applikasjonen vart det òg laga ein brukarmanual (User Manual), denne er skriven på engelsk (vedlegg 11).

Til å lagre oppskriftsfilene laga i RecipeMaker har vi oppretta ein database. BrewingScreen er knytt til denne databasen og hentar ut oppskrifter ved oppstart. Databasen er på ein ekstern server i Nederland.

Vi har laga ei nettsida til prosjektet. Her har vi skrive litt om oss sjølv og om prosjektet og vi har publisert statusoppdateringar, bilete og dokument undervegs.

Tidsmessig har vi brukt ein del fleire timar enn planlagt. Vi kunne nok fint klart å avgrense oppgåva slik at vi heldt den planlagde timebruken, men vi har vert så engasjert og gira i prosjektarbeidet at vi rett og slett valde å ignorere dette litt. Det har ikkje gått ut over fokuset med å kome i mål med eit produkt i tide. Det var planlagt 2000 timar til prosjektarbeidet, 500 timar per person, men den totale timebruken kom opp i 2532 timar.

5.0 Konklusjon

Vi har gjennomført måla definert i forprosjektrapporten, med unnatak av punktet som gjeld kostanadsanalyse og korleis systemet kan gjerast kommersielt. Vi droppa dette punktet då det vart vanskeleg å avgjere ein total pris på anlegget i løpet av prosjektperioden, samstundes som at det vart knapt med tid.

Målet om å byggestyringsskåp vart undervegs i prosjektperioden endra til å lage ein enkel prototype av systemet. Vi valde òg å nytte noko av tida sett av til skåpbygging til å utvide den delen av programmet der oppskriftene skulle skrivast inn.

Vi har kome i mål med eit resultat meir omfattande enn det vi i utgangspunktet hadde planlagt. I tillegg til måla som var sett opp har vi laga ein ekstra applikasjon med tilhøyrande brukarmanual.

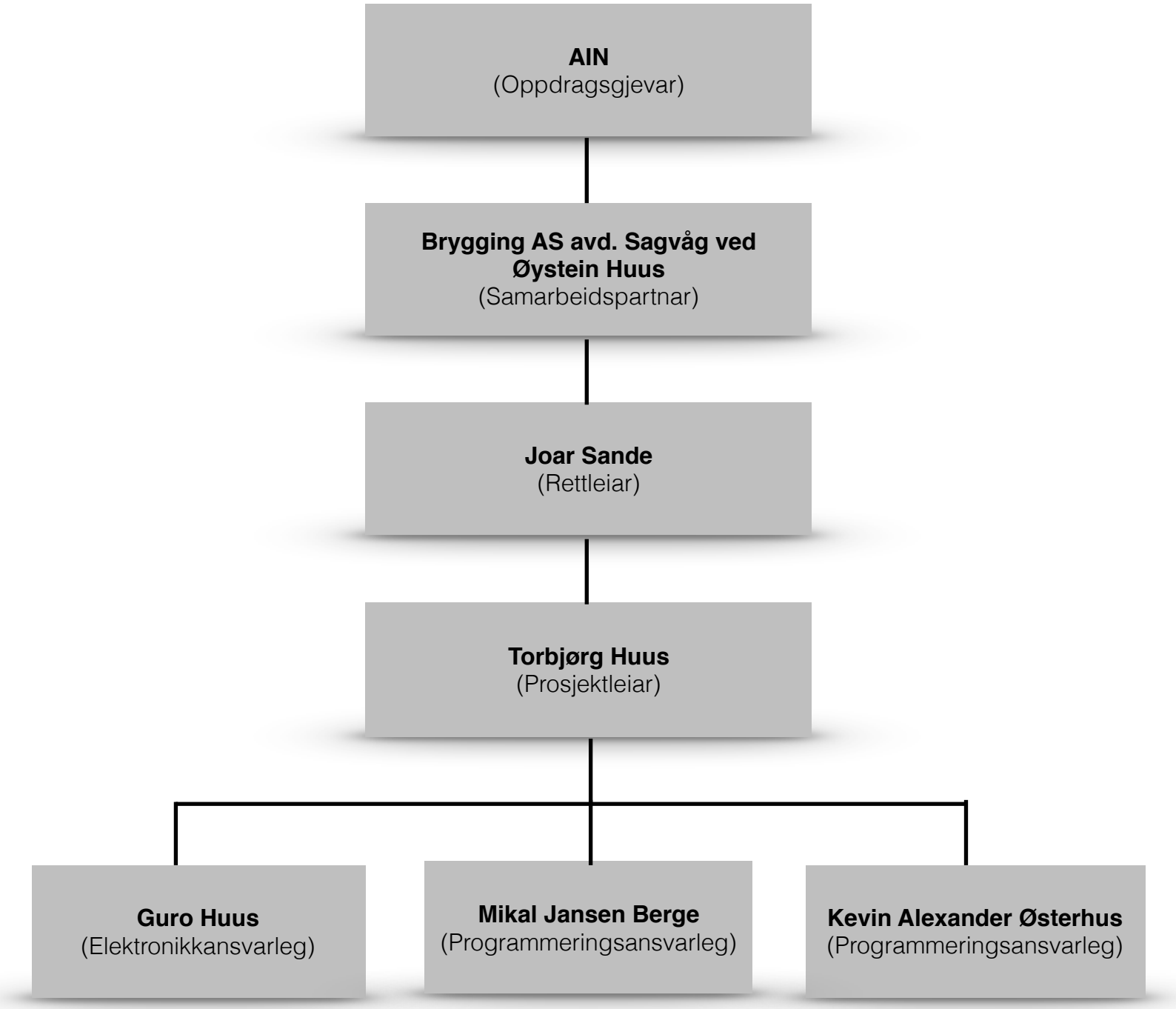
6.0 Drøfting

Gruppa er svært nøgd med det totale produktet og vi har tillært oss mykje kunnskap gjennom prosjektperioden. Vi føler at innsatsen har vore på topp og vi har hatt stor glede i å jobbe med prosjektet. No ser vi fram til vidare samarbeid med Brygging AS avd. Sagvåg, der planen er at vi skal ferdigstille bryggesystemet.

7.0 Prosjektadministrasjon

7.1 Organisering

Organisasjonskartet inneheld beskriving av gruppa sine ulike hovudoppgåver.
Oppdragsgjevaren blir AIN, men vi vil samarbeide med Øystein Huus for Brygging AS avd. Sagvåg (heretter B.S).



Figur 60 Organisasjonskart

Oppdragsgjevar/Samarbeidspartnar:
AIN står som oppdragsgjevar for prosjektet, men sjølve oppgåva er gjort i samarbeid med Øystein Huus for Brygging AS avd. Sagvåg.

Styringsgruppa:
Styringsgruppa består av prosjektansvarleg og rettleiar Joar Sande (HSF) og samarbeidspartnar Øystein Huus (for B.S).

Prosjektgruppa:
Gruppa vår består av 4 avgangsstudentar ved ingeniørstudiet i Førde. Vi har fordelt hovudansvarsoppgåver som vist i organisasjonskartet,

Kontaktinformasjon for prosjektgruppa:

Prosjektgruppa:	**e-post:**	**Telefon:**
Torbjørg Huus	torbjorg.huus@me.com	970 44 633
Guro Huus	guro.huus@me.com	970 42 085
Mikal Jansen Berge	mikjberge@me.com	404 90 376
Kevin Alexander Østerhus	kevin_osterhus@me.com	920 97 136

Rettleiar:	**e-post:**	**Telefon:**
Joar Sande	joar.sande@hisf.no	57 72 26 29 / 414 40 591

Samarbeidspartnar:	**e-post:**	**Telefon:**
Øystein Huus	eysteinhuus@gmail.com	995 91 666

7.2 Gjennomføring i forhold til plan

Møteplan
Vi planla å ha møte kvar 14. dag, men vi var alle einige om at dette er ei rettleiande mengd og at vi vil ta møter etter behov. Gjennomføringa av møter gjekk mykje godt som planlagt med eit snitt på møter kvar 14. dag. Protokoll for desse møta er lagt ved (vedlegg 6).

Milepælar
Undervegs for prosjektperioden var det sett opp nokre viktige milepælar vi måtte jobbe mot.

Fredag 16.01.15	Innlevering av prosjektbeskrivelse
Fredag 13.02.15	Innlevering av forprosjektrapport
Onsdag 08.04.15	Midtvegspresentasjon
Fredag 08.05.15	Innlevering av pressemelding
Fredag 22.05.15	Innlevering av sluttrapport
Onsdag 27.05.15	Presentasjon m/plakat
Fredag 05.06.15	Nettsida ferdigstilt

Prosjektgjennomføring
Undervegs i prosjektet har det vorte nokre endringar i forhold til planlagt aktivitet i gantt-skjemaet. Sjølv om det har vorte nokre endringar frå den opphavlege gjennomføringsplanen har gjennomføringa av prosjektarbeidet gått bra og vi har kome i mål. Vi har laga eit oppdatert gantt-skjema som viser korleis gjennomføringa har gått (sjå vedlegg 4 for planlagt og utført gantt).

Gruppa søkte om utsett innleveringsfrist for forprosjektrapporten av di vi ville avklare nokre detaljar med samarbeidpartner som var vekkreist ei tid. Vi søkte om utsetting til 1. mars. Dette påverka ikkje arbeid med andre planlagte oppgåver i same tidsperiode.

Det har vore mykje endringar i teikningane undervegs som har gjort at arbeidet på desse vart mykje meir enn fyrst planlagt. Desse endringane kjem av at samarbeidspartner har endra ynskje om funksjonaliteten i anlegget. Dette gjorde det vanskeleg å få fullføre komponentlista for bestilling av utstyr, som igjen forseinka det planlagte prosjektarbeidet. Ein annan grunn til at teikningane tok lengre tid, skuldast at arbeidet med å få til ei komplett manuell styring som samstundes kunne virke som ei automatisk styring, var meir komplisert og tidkrevjande enn gruppa i utgangspunktet hadde førestilt seg. Resultatet av dette vart at vi ikkje fekk tid til å byggje styreskåpet, som forklart i neste avsnitt.

Etter midtvegspresentasjonen byrja gruppa å tenke på at vi ikkje ville få moglegheit til å byggje skapet i løpet av prosjektperioden. Dette av di utstyr kom noko seint samstundes som det stadig har vorte endringar i utforminga av systemet. Vi såg at det ville bli knapt med tid til å byggje skapet. Vi drøfta dette med samarbeidspartnar og vart einige om å

utsetje skapbygginga til prosjektet er ferdig. Gruppa har då valt å heller bygge ein prototype.

Vi hadde planlagt at den delen av programmet der oppskriftene skulle skrivast inn skulle vere ein veldig enkel funksjon. Av di det vart litt ventetid på bygging av skap valde vi å bruke meir tid på denne delen av programmet og heller lage ein fullstendig applikasjon, noko samarbeidspartner syns var ein god idé. Med dette aukar vi brukarvenlegheita for denne delen av programmet. Vi kalla applikasjonen for *Recipe Maker* (vedlegg 13). Til denne applikasjonen vart det òg laga ein brukarmanual (User Manual), denne er skriven på engelsk (vedlegg 11).

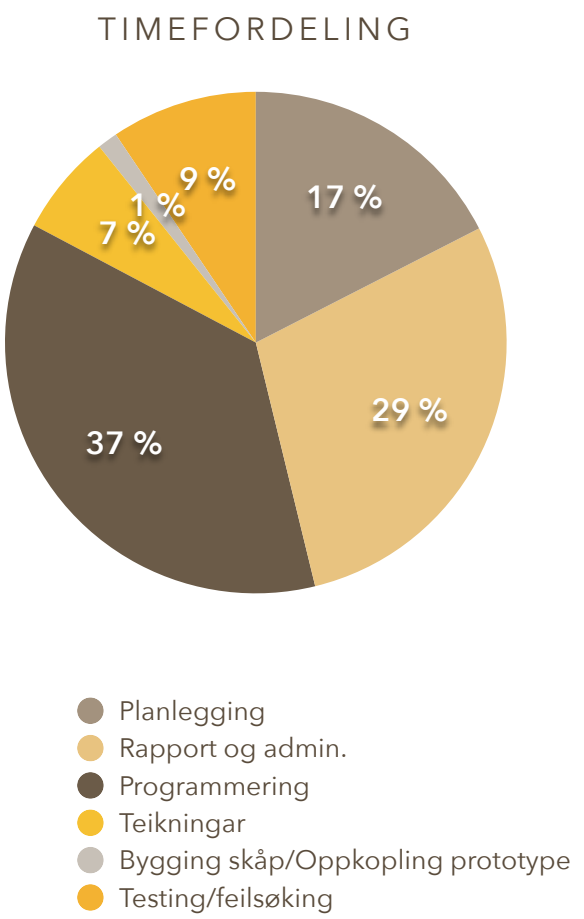
7.3 Tidsressurs

I prosjektgruppa vår har alle valfaget Datamaskiner i nettverk som utgjer 6 timar i veka, vist i tabell 3. Resten av tida vert nytta til prosjektet.

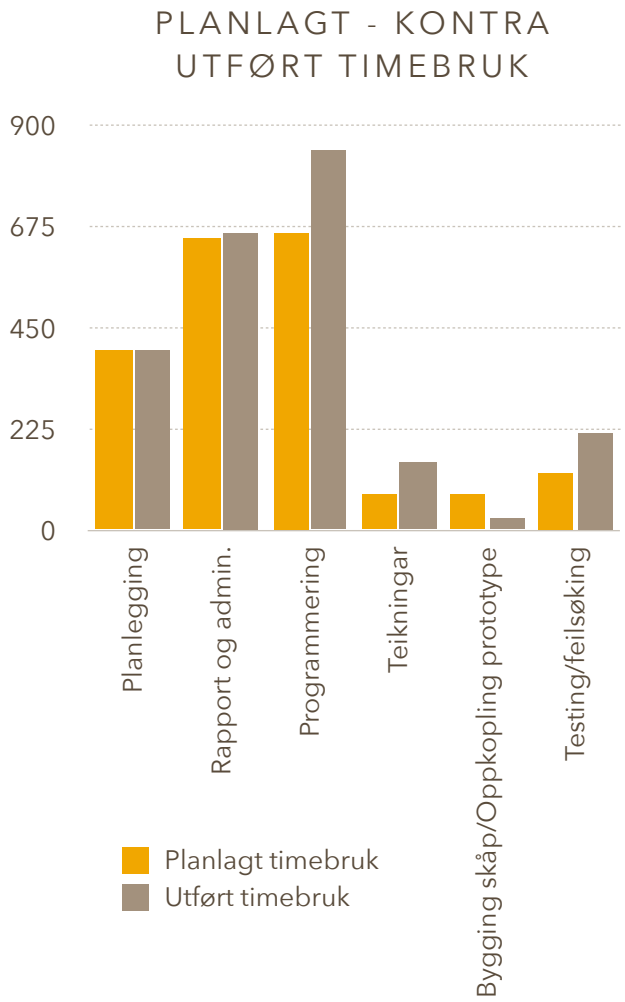
Tid	Måndag	Tysdag	Onsdag	Torsdag	Fredag
08:30-09:15					
09:25-10:10					
10:20-11:05	Datamaskiner i nettverk	Datamaskiner i nettverk			
11:15-12:00	Datamaskiner i nettverk	Datamaskiner i nettverk			
12:30-13:15	Datamaskiner i nettverk				
13:25-14:10	Datamaskiner i nettverk				
14:20-15:05					
15:15-16:00					

Tabell 3 Timeplan

Rettleiande timebruk per elev er sett til ca. 500 timar. Vi har i snitt brukt 633 timar kvar, som utgjer 2 532 timar til saman. På nest side finn du ei oversikt over timebruken fordelt i hovudaktivitetar. Figur 61 viser timefordelinga i prosent, figur 62 viser planlagt timebruk kontra utført timebruk og tabell 4 viser ei oppsummering av timebruken. Ei meir detaljert timeliste for prosjektperioden finn du i vedlegg 5.



Figur 61 Timefordeling i prosent



Figur 62 Planlagt og utført timebruk

OPPSUMMERING TIMEBRUK

Aktivitet	Planlagt timebruk	Guro	Kevin	Torbjørg	Mikal	Utført timebruk
Planlegging	400	88	114	88	114	404
Rapport og admin.	650	291	62	347	64	764
Programmering	660	35	430	35	465	965
Teikningar	80	40	45	21	45	151
Bygging skåp/Oppkopling prototype	80	20	5	0	5	30
Testing/feilsøking	130	27	72	32	87	218
Sum timar	**2 000**	**501**	**728**	**523**	**780**	**2 532**

Tabell 4 Samanlagt timebruk

7.4 Økonomi

Av di gruppa bestemte seg for å lage ein prototype i staden for å bygge skapet, dukka det opp uføresette utgifter i kjøp av materiale. Utanom dette har vi kun hatt utlegg i forbinding med reiser. Høgskulen dekker normalt ikkje meir enn 1000kr i reiseutgifter. Med ein statleg sats på 4,10kr per kilometer[56] betyr det at vi får dekka ca. 244km i reise. Total køyreavstand vi har hatt er 1020km som tilsvarar 4182kr. Då sit vi igjen med ei utgift på 3182kr i køyring.

Tabell 5 viser utgiftene vi har hatt.

Beskriving		Sum
Material frå Farsund Bygg		kr 408,00
Material frå Biltema		kr 294,00
Vasskokar		kr 199,00
Bil Førde-Stord t/r x2		kr 3182,00
Sum totalt		**kr 4083,00**

Tabell 5 Oversikt over utgifter

7.5 Dokumentstyring

Gjennom prosjektarbeidet har vi nytta oss av Dropbox, ei nettskyteneste som synkroniserer filer mellom PC og nett, til fildeling. Her kan vi lagre og dele filer og sidan det vert lagra på nett er sannsynet for å miste dokumenta liten.

Andre verkty vi har nytta er Numbers som er eit reknearkprogram vi har ført timar i, Merlin som har gantt-diagram vi har nytta til å lage framdriftsplan, LucidChart[57] for teikningar og GitHub som vi har brukt til å dele Java-program på nett.

7.6 Nettside

Vi har laga ei nettside for prosjektet vårt som er tilgjengeleg for alle. Her har vi lagt ut ein del dokument kring prosjektarbeidet og statusinnlegg undervegs. Heimesida er laga i Squarespace og domenet har vi kjøpt. Squarespace er eit nettbasert publiseringssystem, eit sokalla Content Management System (CMS), som tilbyr enkel publisering og handtering av nettsider og bloggar, og attpåtil tenester for lagring av data på verdsveven.

Lenkje til heimesida: www.autobrygg.no

8.0 Figur- og tabelliste

Figurar

Tabellar

9.0 Referanseliste

[1] Studiehandbok HISF: http://studiehandbok.hisf.no/no/content/view/full/13996 (21.05.15, 21:00)
[2] Ølhistorie: http://www.hansaborg.no/no/produkter/ol/historie/ (21.05.15, 21:20)
[3] Ølhistorie: Horne, Thomas og Eick, Colin, 2013. *Øl brygging fra hånd til munn.* Oslo: Kagge Forlag AS. 2. utg. (side 10-12)
[4] Tilleggskort: https://www.abelectronics.co.uk/products/17/raspberry-pi-plus/ (21.05.15, 21:30)
[5] Datablad: http://auberins.com/images/Manual/SMD100_Manual.pdf (21.05.15, 21:40)
[6] PID-regulator: http://www.bitbok.no/kjemiprosess/boka/3-reguleringsteknikk/pid-regulator/ (21.05.15, 21:40)
[7] Timer: http://auberins.com/images/Manual/JSL-73B.pdf (21.05.15, 21:45)
[8] CR5 01: http://www.ebay.com/itm/New-DC5V-1-2-to-1-1-4-Electric-Ball-Valve-Motorized-Ball-Valve-/121414316485?var=420350788960 (21.05.15, 21:50)
[9] SSR: http://www.auberins.com/SSR%20Series-RS1A.pdf (21.05.15, 21:55)
[10] SSR: http://electronics-diy.com/electronic_schematic.php?id=956 (21.05.15, 21:55)
[11] SSR: http://www.allaboutcircuits.com/vol_4/chpt_5/5.html (21.05.14, 21:55)
[12] SSR: http://auberins.com/index.php?main_page=page&id=10&chapter=0 (21.05.15, 22:10)
[13] Youtube: https://www.youtube.com/watch?v=yM_5tGAZD64 (21.05.15, 22:12)
[14] RPi: https://www.raspberrypi.org/about/ (21.05.15, 22:15)
[15] I2C: http://www.totalphase.com/support/articles/200349156 (21.05.15, 22:30)
[16] ADC Pi Plus: http://ww1.microchip.com/downloads/en/DeviceDoc/22088b.pdf (21.05.15, 22:35)
[17] ADC Pi Plus: https://www.abelectronics.co.uk/products/17/Raspberry-Pi/56/ADC-Pi-Plus---Raspberry-Pi-Analogue-to-Digital-converter (21.05.15, 22:35)
[18] IO Pi Plus: https://www.abelectronics.co.uk/products/17/Raspberry-Pi--Raspberry-Pi-2-Model-B/54/IO-Pi-Plus (21.05.15, 22:45)
[19] RTD-element: Larsen, Bjørnar, 2002. *Industriell måleteknikk*. Oslo: Vett & Viten AS. 2. utg. (side 232-233)
[20] LC RT4: http://www.lucid-control.com (21.05.15, 22:55)
[21] LC RT4: http://www.lucid-control.com/support/downloads/ (21.05.15, 22:55)
[22] PT-1000: http://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/A707D00EE0F558D6C12574E1002C2D1C/$file/tsiec751_ce.pdf?OpenElement (21.05.15, 23:00)
[23] PT-1000: http://www.ra.danfoss.com/TechnicalInfo/Literature/Manuals/04/IC.PD.P30.I3.02.pdf (21.05.15, 23:00)
[24] Kapasitiv målar: Larsen, Bjørnar, 2002. *Industriell måleteknikk*. Oslo: Vett & Viten AS. 2. utg. (side 197)

[25] Ultralyd: Larsen, Bjørnar, 2002. *Industriell måleteknikk*. Oslo: Vett & Viten AS. 2. utg. (side 194)
[26] Ultralyd: http://www.tu.no/automatisering/automatiseringsakademiet/kategorier/maleteknikk-og-analyse/nivamaling/2013/03/27/universell-ultralyd-12 (21:05.15, 23:10)
[27] MPX5010: http://www.retelektronika.hu/DataSheets/50_SZENZOROK/MOT_005/MPX5010.pdf (21.05.15, 23:15)
[28] Raspbian: https://www.raspbian.org (21.05.15, 23:20)
[29] RPi download: https://www.raspberrypi.org/downloads/ (21.05.15, 23:25)
[30] RPi Teach: https://www.raspberrypi.org/documentation/installation/installing-images/README.md (21.05.15, 23:25)
[31] SSH: http://en.wikipedia.org/wiki/Secure_Shell (21.05.15, 23:25)
[32] SSH: https://www.raspberrypi.org/documentation/remote-access/ssh/passwordless.md (21.05.15, 23:25)
[33] pi4j: http://pi4j.com (21.05.15, 23:30)
[34] Statisk IP-adresse: http://www.modmypi.com/blog/tutorial-how-to-give-your-raspberry-pi-a-static-ip-address (21.05.15, 23:30)
[35] Boot script: https://raspberrypi.stackexchange.com/questions/8734/execute-script-on-start-up/8735#8735?newreg=8cfd083afdb1418a9254cbdbb8a5040d (21.05.15, 23:30)
[36] Youtube: https://www.youtube.com/watch?v=4XpnKHJAok8 (21.05.15, 23:30)
[37] Git: http://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (21.05.15, 23:30)
[38] Merging and branching: https://git-scm.com/about/branching-and-merging (21.05.15, 23:30)
[39] Branching: http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#_git_branching (21.05.15, 23:30)
[40] commit: http://git-scm.com/docs/git-commit (21.05.15, 23:30)
[41] push: http://git-scm.com/docs/git-push (21.05.15, 23:30)
[42] pull: http://git-scm.com/docs/git-pull (21.05.15, 23:30)
[43] clone: http://git-scm.com/docs/git-clone (21.05.15, 23:30)
[44] branch: https://git-scm.com/about/branching-and-merging (21.05.15, 23:30)
[45] merge/rebase: https://www.atlassian.com/git/tutorials/merging-vs-rebasing/workflow-walkthrough/ (21.05.15, 23:30)
[46] GitHub: http://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/ (21.05.15, 23:40)
[47] Iterasjonsmodellen: http://en.wikipedia.org/wiki/Iterative_and_incremental_development (21.05.15, 23:40)
[48] API: http://www.lucid-control.com/support/downloads/ (21.05.15, 23:40)
[49] pi4j: http://pi4j.com (21.05.15, 23:40)
[50] pi4j: http://pi4j.com/apidocs/index.html (21.05.15, 23:40)
[51] MVC-modell: http://no.wikipedia.org/wiki/Model–view–controller (21.05.15, 23:40)
[52] Java9: http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-March/001981.html (21.05.15, 23:40)

[53] Scene builder: http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html (21.05.15, 23:40)
[54] CLI: http://en.wikipedia.org/wiki/Command-line_interface (21.05.15, 23:40)
[55] Angelas blog: https://blogs.oracle.com/acaicedo/entry/managing_multiple_screens_in_javafx1 (21.05.15, 23:40)
[56] Regjeringa: https://www.regjeringen.no/nb/dokumenter/kostgodtgjorelse-og-nattillegg/id438637/ (21.05.15, 23:40)
[57] LucidChart: https://www.lucidchart.com/personahomepage?utm_source=google&utm_medium=cpc&utm_campaign=lucidchart_norway&gclid=CP3g-Yvg0cUCFev3cgodyk8ADg (21.05.15, 23:50)

10.0 Vedlegg

*Sjå eiga mappe for vedlegg

Vedlegg 1: Fullmaktsskjema
Vedlegg 2: Prosjektbeskrivelse
Vedlegg 3: Forprosjektrapport
Vedlegg 4: Gantt
Vedlegg 5: Timeliste
Vedlegg 6: Protokoll
Vedlegg 7: Pressemelding
Vedlegg 8: Epost
Vedlegg 9: Komponentliste
Vedlegg 10: Datablad
Vedlegg 11: User Manual (RecipeMaker)

Vedlegg som vert levert på minnepenn:

Vedlegg 12: Teikningar
Vedlegg 13: Programvare