Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Jone Vassbø

Dronebasert sanking av sau

Masteroppgave i Datateknologi
Veileder: Svein-Olaf Hvasshovd
Juni 2023

NTNU
Kunnskap for en bedre verden

Jone Vassbø

Dronebasert sanking av sau

Masteroppgave i Datateknologi
Veileder: Svein-Olaf Hvasshovd
Juni 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU
Kunnskap for en bedre verden

Sammendrag

For år 2020 rapporterte Mattilsynet i sin årsrapport at det var 1,9 millioner sauer og lam på beite i utmark i Norge [17]. Kombinert med tall fra Statistisk Sentralbyrå fra samme år, gir en overslagregning et gjennomsnitt på 137 sauer/lam per jodbruk [2]. Disse dyrene slippes hver vår i utmarken, oftest uten veiforbindelse. Når høsten kommer sankes dyrene inn ved at mennesker går til fots gjennom utmarken, opptil flere ganger. Dette er en tid- og energikrevende jobb, som repeteres hvert år.

På bakgrunn av denne omfattende prosessen ved å sanke hjem sau, er denne oppgaven utformet. Oppgaven går ut på å bytte menneskene ut med 4 droner som kontrolleres av en algoritme. Målet med oppgaven er å finne ut i hvilken grad dette er mulig. Oppgaven går dypere inn på hvordan algoritmen kan håndtere gjeting langs en sti, hente en flokk fra utmark og inn på sti, samt gjeting av sauene gjennom et stikryss.

Det er utformet en kontrollalgoritme bestående av flere moduler, deriblant en banekoordinator, punktkalkulator, og en generell kalkulator. For å få de forskjellige modulene til å jobbe sammen på de rette tidspunktene, er det satt opp en tilstandsmaskin som kontrollerer helheten.

Oppgaven har resultert i kontrollalgoritme, simulering av droner, sauer og gjeteområde, samt en visualisering som presenterer gjetingen gjennom et brukergrensesnitt.

På bakgrunn av omfattende både manuell og automatisk testing, med blant annet 1584 ende til ende tester, er det konkludert med at den menneskelige sankingen av sau kan i høy grad erstattes med 4 droner som kontrolleres av en algoritme.

Abstract

In year 2020, the Norwegian Food Authority reported 1,9 million sheeps and lamb where released to graze in outfield pastures in Norway [17]. Combined with numbers from the Central Statistical office, an estimates shows that there are in average 137 sheeps/lambs per farm [2]. These animals are released every spring to a outfield, usually wihtout any road connection. When autumn comes the animals are gathered in by people walking through the outfield, up to several times. This is a time- and energy-consuming job, which is repeated every year.

On the basis of this extensive process of herding home the sheeps, this thesis has been designed. The goal is to replace the humans with 4 drones that are controlled by an algorithm. The aim of the thesis is to find out to what extent this is possible. The thesis goes deeper into how the algorithm can handle herding along a path, fetching a flock from the open field and onto a path, as well as herding the sheep through a path junction.

A control algorithm consisting of several modules has been designed and developed, including a path coordinator, point calculator and a general calculator. In order to make the different modules work together at the right times, a state machine has been set up that controls the whole.

The thesis has resulted in a control algorithm, simulation of drones, sheep and herding area, as well as a visualization that presents the herding through a user interface.

On the basis of extensive both manual and automatic testing, including 1584 end-to-end tests, it has been concluded that the human herding of sheep can to a large extent be replaced with 4 drones controlled by an algorithm.

Forord

Denne rapporten er skrevet som en masteroppgave på studiet Datateknologi med hovedprofil Programvaresystemer ved Institutt for datateknologi og informatikk, Fakultetet for informasjonsteknologi og elektroteknikk ved Norges teknisk-naturvitenskapelige universitet i Trondheim.

I forkant av oppgaven ble det utført et forprosjekt. Prosjektet fokuserte på tilegning av domenekunnskap og utvikling av programvare. Denne masteroppgaven har tatt utgangspunkt i forprosjektet og videreutviklet det. Valgene som ble tatt og konklusjonene som ble funnet er blitt videreført, videreutviklet og dokumentert i denne rapporten.

Oppgaven resulterte i en programvare som inneholder en kontrollalgoritme for automatisk sanking av sauer ved hjelp av fire droner. Kontrollalgoritmen ved hjelp av dronene er i stand til å erstatte store deler av kompleksiteten ved sanking av sau og derfor erstatte menneskene i denne prosessen. Algoritmen kan testes gjennom brukergrensesnittet publisert på https://sau.vassbo.as. Videoer når algoritmen kjører er publisert på https://bit.ly/gjeting-liste. Kildekoden ligger åpent på https://github.com/jonev/sheep-herding.

Ønsker å rette en takk til veileder og domenekspert Professor Svein-Olaf Hvasshovd for veiledning under utførelsen av oppgaven.

Jone Vassbø

Trondheim, 09.06.2023

Innhold

Figurer

Tabeller

Liste av kodebiter

Definisjoner

ASP.NET	Rammeverk for bygging av web applikasjoner og servicer med .NET og C# [3].
C#	Programmeringsspråk [7].
Dronene	Oversiktsdronen og de tre gjeterdronene.
Gjeterdrone	Drone som får kommandoer fra Oversiktsdrone og påvirker sauenes retning.
Oversiktsdrone	Drone som kontrollerer gjetingen og ikke påvirker sauene direkte.
Utspringersau	En sau som er på vei ut av flokken.
Vilkårlighetsvinkel	Vinkel ut fra gjeteretning en sau kan løpe mot under gjeting, for å introdusere vilkårlig oppførsel hos sauene.
.NET	Plattform med åpen kildekode fra Microsoft for bygging av applikasjoner [1].

1 Rapportens oppbygging

Kapitel 1 - Innledning inneholder bakgrunnen for oppgaven, hvilken problemstilling det forskes på og hvilke avgrensninger som er satt.

Kapitel 2 - Tidligere arbeid inneholder utdrag fra artikler funnet med høyest relevans til denne oppgaven.

Kapitel 3 - Metode forklarer hvordan kontrollalgoritmen, simuleringen og visualiseringen fungerer hver for seg og sammen som en komplett programvare, hvilke valg som er tatt og hvorfor.

Kapitel 4 - Resultat presenterer hvordan kontrollalgoritmen yter i de forskjellige scenariene den er testet og at funksjonaliteten rundt posisjonering av gjetere fungerer.

Kapitel 5 - Diskusjon diskuterer metodene som ble presentert i kapitel 3 - Metode og hvordan metodene har utviklet seg i flere iterasjoner.

Kapitel 6 - Konklusjon presenterer i hvilken grad det konkluderes med at problemstillingen og tilhørende forskningspørsmål kan håndteres av kontrollalgoritmen.

Kapitel 7 - Videre arbeid presenterer hvilket arbeid som er relevant for videreutvikling av kontrollalgoritmen.

2 Innledning

I dette kapittelet er det forklart hvilken problemstilling oppgaven forsøker å løse, hva som er bakgrunnen for oppgaven, hvilke avgrensninger det er satt og hvilken forskningsmetode som er brukt.

2.1 Bakgrunn

Mattilsynet rapporterer i sin årsrapport for 2021, at det i 2020 var 1,9 millioner sauer og lam på beite i utmark [17]. Statistisk sentralbyrå melder samme år at det finnes 13 807 jordbruksbedrifter som driver med sau over ett år [2]. En overslagsregning gir et gjennomsnitt på 137 sauer/lam per jordbruk.

For å spare fôr samtidig som man holder vegetasjonen nede, slipper bønder landet rundt sauene sine på utmark hver vår. Utmark kan defineres som opp i fjellet, eller ut på vidda. En gjenganger er et stort område, som gjerne eies av flere bønder, og som ikke er dekket med veiforbindelse. Når høsten kommer er bonden nødt til å dra opp i utmarken å sanke sauene hjem. Det gjøres som oftest til fots. Dette er et omfattende arbeid hvor bonden ofte får hjelp av venner, familie og spesialtrente hunder. Man går da på kryss å tvers av hele utmarken slik at man får se over og få med seg alle sauene hjem. Det er skjeldent, om ikke aldri, at det holder å ta én tur opp i marka for å sanke sauene hjem. Utmarken er også oftest så stor at man ikke klarer å se over hele området på én tur.

Med et gjennomsnitt på 137 sauer/lam per jordbruk er dette ett tids- og energikrevende arbeid som repeteres hvert år. På bagrunn av disse tallene finnes det motivasjon for å automatisere sankingen av sau.

2.2 Problemstilling og mål

Denne oppgaven er utformet for å utfordre måten sauer sankes på i dag, beskrevet i kapitel 2.1. I steden for at mennesker skal gjete sauer, er det i denne oppgaven forsket på å bruke droner. Det er forhåndsdefinert fra oppgavegiver at det skal brukes fire droner. En oversiktsdrone og tre gjeterdroner. Oversiktsdronen har i oppgave å ha kontroll på hvor sauene er og drive gjetingen ved å sende kommandoer til gjeterdronene. Gjeterdronene flyr i en høyde hvor sauene blir skremt av de. De har i oppgave å drive sauene i rett retning. Gjeterdronene jobber utelukkende på kommandoer fra oversiktsdronen. Disse kommandoene inneholder koordinater til hvor gjeterdronene skal fly.

En flokk på fem sauer skal gjetes fra et vilkårlig sted i utmarken og inn til gitt målposisjon. Målposisjon vil typisk være et sted i forbindelse med utmarken som har veiforbindelse. Da kan bonden laste sauene inn i en henger/lastebil og kjøre dem til gården for vinteren. For at gjetingen skal oppleves mest mulig naturlig for sauene, legges den opp i forbindelse med turstier i utmarken.

Dette gjøres ved at bonden definerer en bane sauene skal gjetes etter. Dronene skal fly den korteste veien til saueflokken å gjete dem inn via denne predefinerte banen.

Oppgaven går ut på å lage en simuleringsmodell for gjetingen av sau. Dette innbærer å lage kontrollalgoritmen som skal avgjøre hvordan gjeterdronene skal bevege seg. Det lages også et brukergrensesnitt som visualiserer hvordan gjetingen foregår og en programvare som kjører simuleringen og kontrollalgoritmen.

Ut i fra denne oppgavebeskrivelsen er det utformet følgende problemstilling:

- **I hvilken grad kan menneskelig sanking av sau erstattes med fire droner kontrollert av en algoritme?**

Den overordnede problemstillingen omhandler mange deler kompleksitet når det kommer til å erstatte en nåværende menneskelig operasjon. På bakgrunn av dette er det utformet følgende forskningspørsmål:

F1: I hvilken grad kan kompleksiteten ved å gjete en flokk sauer langs en sti håndteres av en kontrollalgoritme?

Ved gjeting langs en sti er dronene nødt til å følge stiens naturlige bane. Dette betyr at de må posisjonere seg slik at ved for eksempel svinger, får sauene en naturlig bane hvor de holder seg på stien. Det må også tas høyde for å gjøre justeringer om en sau/saueflokken velger å forlate stien.

F2: I hvilken grad kan kompleksiteten ved å gjete en flokk sauer fra utmarken og inn på en sti håndteres av en kontrollalgoritme?

Ved oppstart av gjeting er det høy sannsynlighet for at sauene står utenfor en sti å gresser. Første operasjon er å gjete sauene inn på stien. Sauene gjetes fra deres utgangspunkt og inn på nærmeste punkt på stien uten å ta unødige omveier.

F3: I hvilken grad kan kompleksiteten ved å få en flokk sauer til å velge rett utgang i et stikryss håndteres av en kontrollalgoritme?

Ved gjeting langs en sti vil man møte på stikryss. Om stikrysset har to utganger finnes det 50% sjanse for at sauene går til feil utgang. Denne kompleksiteten må håndteres av dronene slik at sauene føres til rett utgang.

Hovedmålet ved oppgaven er å løse så mye av kompleksiteten sanking av sau innebærer ved hjelp av fire droner og en kontrollalgoritme som er delt opp i forskningsspørsmålene over.

2.3 Avgrensninger

I dette kapittelet er avgrensningene til oppgaven utdypet. Oppgaven modellerer deler av den fysiske verden, det er derfor presisert hva som er modellert og hva som er utelatt. Grunnlaget for avgrensningene er basert på domenekspert og veileders kunnskap rundt sanking av sau i norskt terreng, samt begrensning av oppgavens omfang.

Oppgavens fokus er kontrollalgoritmen, den er derfor avgrenset til å virtualisere terrenget, sauene og dronene. Dette betyr at denne oppgaven omhandler utviklingen av programvare. Det er ikke involvert noe fysisk utstyr eller sauer.

Oppgaven er avgrenset til sanking av èn flokk med fem sauer som befinner seg i umiddelbar nærhet av hverandre. Dronene sendes ut til å sanke flokken fra dens posisjon frem til en måldestinasjon. Banen som skal følges er predefinert og er lagt opp til å følge terrengets stier. Bonden må selv sette opp banen og det fordrer derfor at hen er kjent i terrenget. Når èn flokk er sanket må bonden repetere prosessen for å sanke neste flokk.

Det er fastsatt tre droner som skal gjete sauene, det vil si skremme dem til å gå i riktig retning, disse er kalt gjetere. En fjerde drone vil fly så høyt over saueflokken at den ikke påvirker sauene, denne er kalt oversiktsdrone. Når det refereres til dronene, er det snakk om alle fire dronene. Oversiktsdronen vet til en hver tid posisjonen til alle sauene og gjeterne. Denne dronen vil eksekvere kontrollalgoritmen. Kontrollalgoritmen vil produsere kommandoer ut i fra hvor dronene og sauene er. For hver kjøring av kontrollalgoritmen vil den produseres fire kommandoer, èn til hver drone.

Grensesnittet mellom kontrollalgoritmen og dronene er kommandoer med koordinater dronene skal fly til. Det er tiltenkt at hver drone må ha lokal styring som tar i mot koordinatet og styrer dronen til å fly til anvist koordinat. Om en drone ikke mottar en kommando, vil den stå i ro på forrige koordinat den mottok. Dronene må ha innebygget styring for å unngå å støte i hverandre eller sauene. Skulle en drone befinne seg på samme koordinat som en annen drone eller en sau, er det antatt at de er på forskjellige høyder.

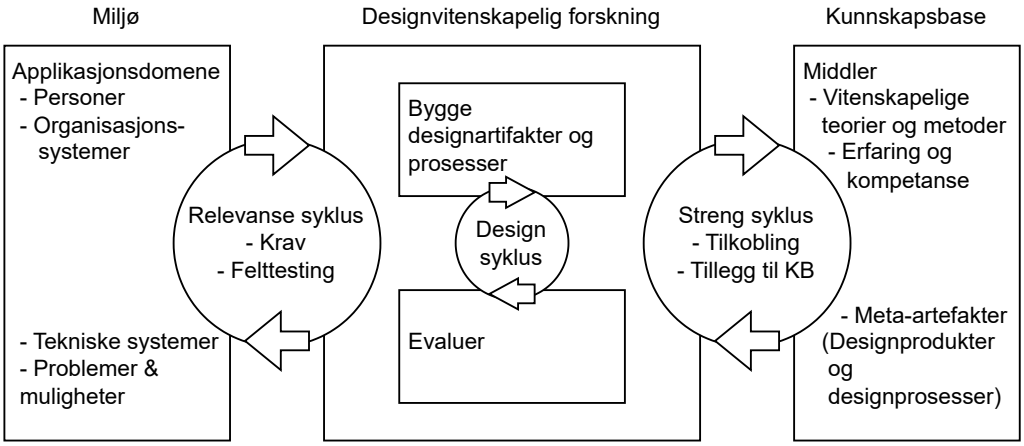
Terrenget sauene gjetes i er forenklet til å være to dimensjoner. Det vil si at det finnes kun et flatt område uten høydeforskjeller, trær eller steiner. Vær og vind påvirker ikke dronene. Det vil ikke oppstå uventede forstyrrelser i gjetingen, som for eksempel at sauene blir skremt av noe. Det antas at kommunikasjonen mellom dronene alltid fungerer.

Høyden gjeterene flyr på er antatt å være den optimale høyden for å skremme sauene i ønsket retning og potensiell endring i høyden er abstrahert bort.

Stikryss er forenkelt til å ha èn inngang og to utganger. Sauene satt opp til å alltid ta utgangen til venstre, mens dronene ønsker å gjete sauene til høyre utgang.

2.4 Forskningsmetode

Oppgaven går ut på å utvikle en programvare med hovedfokus på kontrollalgoritme, men også simulering av sauer og visualisering av gjetingen. Oppgaven inneholder ingen føringer på hvordan selve programvaren skal utvikles eller hvordan den skal se ut. Med utgangspunkt i disse forutsetningene er det brukt en høyst iterativ forskningsmetode med utgangspunkt i "A Three Cycle View of Design Science Research" [12]. Figur 1 er hentet fra artikkelen [12] og visualiserer de forskjellige syklusene. Denne oppgaven legger størst vekt på design syklusen. Programvaren utvides i små iterasjoner og for hver iterasjon evalueres endringen med både manuelle og automatiske tester. Ved bestemmelser for programvarens retning og overordnet fokus brukes relevanse syklusen, hvor det er gjort hyppige intervjuer med veileder og domenekspert. For inspirasjon og avsjekk er tilkoblingen mot tidligere arbeid brukt, hvor flere artikler er undersøkt.



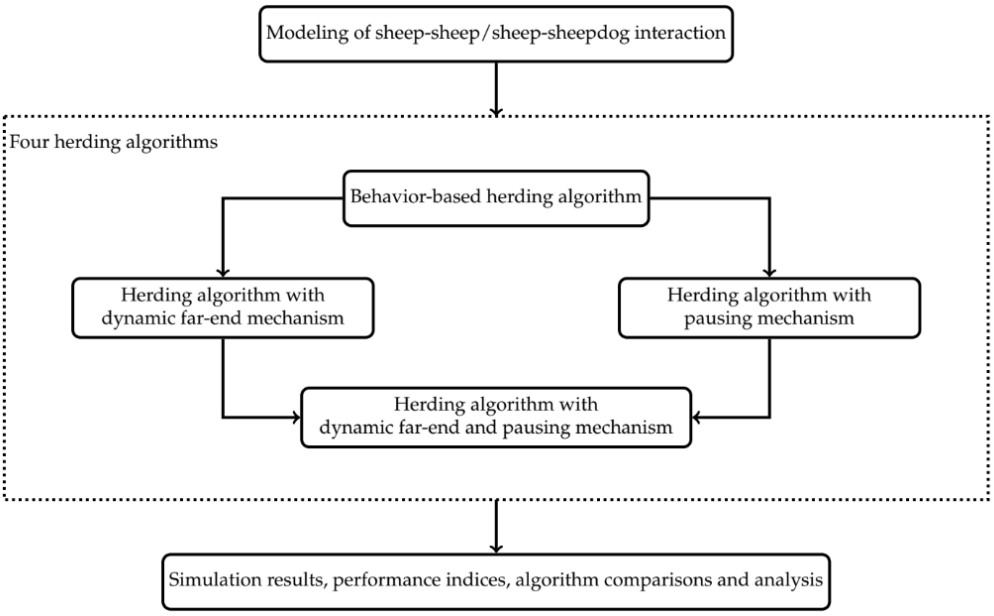
Figur 1: Designvitenskapelig forskning - Tre sykluser [12].

3 Tidligere arbeid

I dette kapittelet er tre artikler med tidligere arbeid lagt frem og relevansen til arbeidet er diskutert. Oppsummering finnes i siste delkapittel.

3.1 Behavior-based herding algorithm for social force model based sheep herd

En gruppe fra Kina har gjort en studie og publisert en rapport i januar 2023 [8]. De har satt opp fire algoritmer. Den første algoritmen brukes som fundamentet, deretter bygges det på med to forskjellige algoritmer. Til slutt kombineres alle fire algoritmene. Oppbyggningen er visualisert i figur 2. Algoritme èn er "oppførsel basert gjeting", med denne som fundament er det bygget to algoritmer, "dynamisk sluttpunk" og "pause mekanisme". Til slutt kombineres alle tre algoritmene til en fjerde; "oppførsel basert gjeting med dynamisk sluttpunk og pause mekanisme".



Figur 2: Oppbygging av algoritmene [8].

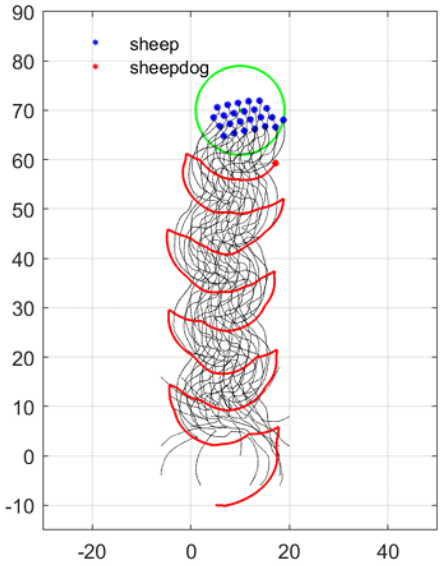
Studien tar utgangspunkt i matematisk modellering av bevegelsene til en gjeter-hund som fører saueflokken slik det er visualisert i figur 3. Utgangspunktet til modelleringen er hunden og hundens synsvinkel. Hunden er på samme høyde som sauene og vil derfor kun se den første rekken med sauer.

Den oppførselsbaserte gjetingen deres tar utgangspunkt i å bevege hunden med en side til side formasjon. Side til side bevegelsen blir definert ut i fra et sett med fire "kritiske" sauer. Enkelt forklart er disse fire sauene de som befinner seg i ytterkantene forran og bak i flokken, samt i synsvinkelen til hunden. Valgene av disse fire kristiske sauene ble initielt utarbeides i deres tidligere

arbeid [16], men matematikken er tatt med i rapporten [8].

Dynamisk sluttpunkt mekanismen tar hånd om utfordringen man får når sauene når målposisjonen, men det er ingenting som holder dem der. Dette løses ved å ha et dynamisk sluttpunkt som ligger på en sirkel som er slått rundt målposisjonen og som flytter seg avhengig av vinkelen sauene har på målposisjonen.

Pause mekanismen er implementert for å replikere hundens måte å gjete på. Med en stor saueflokk blir det store avstander for hunden å løpe. Denne er implementer for å spare energi, på samme måte som for hunden. Til slutt er disse tre algoritmene kombinert til en fjerde. Algoritmene måles mot hverandre med antall sauer på 10, 24 og 50. Resultatene viser at algoritme fire presterer best på flere områder og at det vil ta lengre tid ettersom størrelsen på saueflokken øker.



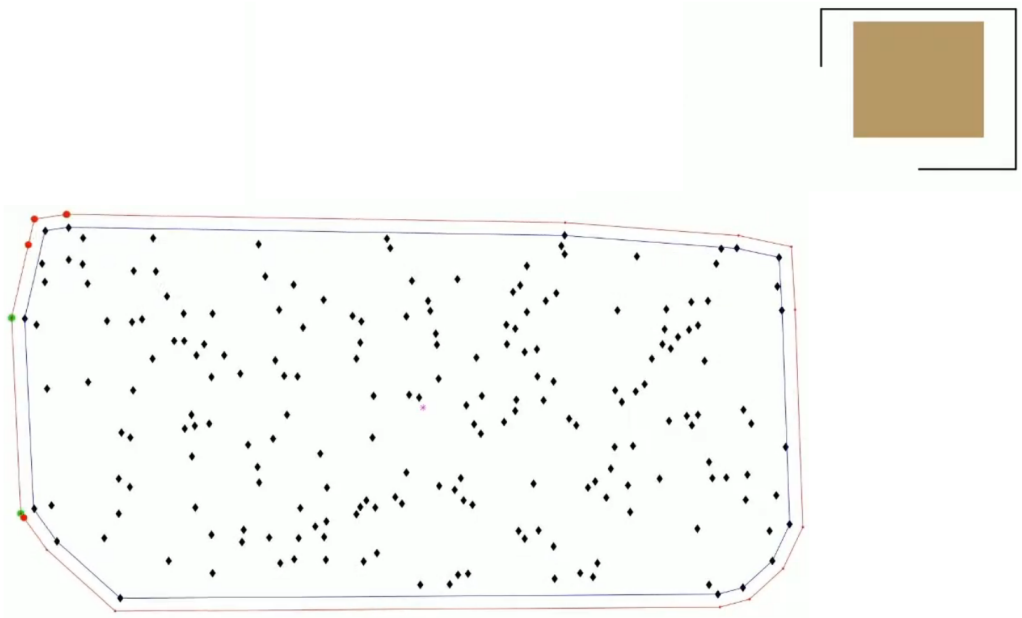
Figur 3: Gjeting av sauer basert på matematisk modellering av hund [8].

Konseptene rundt algoritmen brukt i denne studien [8] er relevant for denne oppgaven. Spesielt når det kommer til gjeting fra et punkt til et annen som en del av banen. De har tatt utgangspunkt i en større flokk med sauer, men det har ikke stor innvirkning på konseptene. De har også brukt kun en drone til gjeting, mens det i denne oppgaven er brukt tre droner. Det har noe å si på den kontinuerlige "side til side" bevegelsen, men konseptet har skapt inspirasjon til håndtering av utspringersau. Med tanke på investeringer vil det være en økonomisk fordel å bruke èn drone, i steden for fire. På den andre siden, er kompleksiteten på bevegelsene dronen gjør høyere. Studien tar utgangspunkt i hundens synsvinkel, hvor det i denne oppgaven er brukt synsvinkelen til en oversiktsdrone. Hunden vil bare se første rekke med sauer, mens oversiktsdronen vil til en hvert tid vite koordinatene til alle sauene. Det er derfor fundamentalt forskjellig utgangspunkt når man skal avgjøre hva som blir den neste bevegelsen i gjetingen.

3.2 Robotic herding of farm animals using a network of barking aerial drones

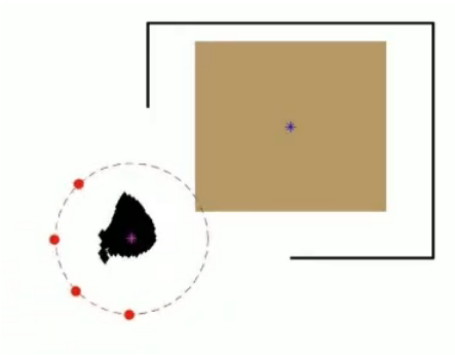
I 2022 har en gruppe fra Australia gjort en studie [15] på sanking av sauer ved bruk av en sverm med bjeffende droner. Rapporten fokuserer på en matematisk modell for effektiv kontrollering av dronene under samling av mellom 200 til 1000 sauer på et åpent område. Deriblant utdypes matematikken rundt bevegelsen til dronene når det kommer til posisjonering, håntering av hjørner og hvordan bremsing skal foregå under samling. Når sauene er samlet brukes det en annen modell til å gjete de i en rett linje inn i en innhengning. Figur 4 og 5 er hentet fra en video [11] publisert sammen med rapporten [15].

Figur 4 visualiserer hvordan dronene samler flokken av sauer. De flyr i et mønster langs de røde linjene, som er nært nok til å skremme sauene. Etter hvert som sauene trekker seg sammen blir området inni de røde linjene mindre. Denne samlingen foregår helt til flokken er samlet i henhold til figur 5 og klar til gjeting. Sauene skal gjetes til innhegningen oppe til høyre i figuren som er visualisert med svart ramme og brun bakgrunn. Matematikken bak samlingen av sauene er fokuset i rapporten [15].



Figur 4: Samling av droner på åpent område, før gjeting mot innhengning oppe til høyre [15].

Figur 5 visualiserer hvordan dronene gjeter flokken fra det åpne området og inn i innhengningen. Dette er etter at algoritmen fra figur 4 har samlet flokken. For å gjete flokken, legger dronene seg på en sirkel rundt flokken og beveger seg frem og tilbake på sirkelen, for å forhindre at sauer slipper gjennom. Denne bevegelsen kan beskrives som at dronene roteres frem og tilbake på bakssiden av flokken, sett fra målposisjonen. Samtidig som sauene ikke slipper unna, drives de mot målposisjonen.



Figur 5: Gjeting av droner mot innhengning [15].

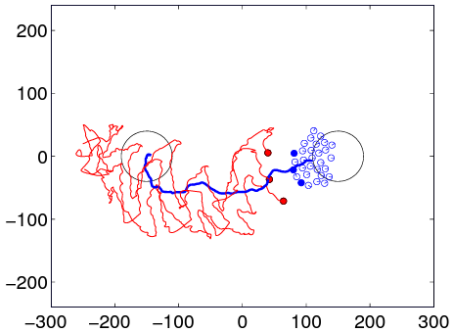
Som nevnt i kapittel 2.3 er denne oppgaven avgrenset til å gjete fem sauer i nærheten av hverandre. Algoritmen fra rapporten [15] som omhandler å samle sauene til en flokk er derfor lite relevant. Algoritmen for å gjete sauene til en målposisjon etter at de har blitt samlet er dog mer relevant. De har ikke tatt med kompleksiteten ved å gjete sauene langs en bane med vinkler, men måten dronene sørger for at ingen sauer slipper unna er til inspirasjon for denne oppgaven.

3.3 Autonomous shepherding behaviors of multiple target steering robots

I 2017 ble det gjort en studie [14] i Korea. Studien ligner på de to forrige studiene som er utdypet tidligere i dette kapittelet. Det legges frem en matematisk modell for både samling og gjeting med en til fem gjetere, av en flokk på 20, 60 eller 100 sauer. Når sauene er samlet gjetes de på en rett linje fra startposisjon til målposisjon. Figur 6 er hentet fra studien. De røde prikkene er gjetere og de blå er sauer. Strekene som vises er banene gjeterne og senter av flokken har fulgt gjennom gjetingen.

Gjeterne følger en algoritme hvor så lenge det er flere saueflokker enn gjetere, samles flokkene. Da blir hovedflokken kun holdt på samme sted og ikke gjetet mot målposisjon. Når antall flokker er mindre enn gjetere, gjetes hovedflokken mot målposisjon samtidig som resterende flokker samles mot hovedflokken. Når sauene når målposisjon går gjeterne over til å holde flokken på samme sted igjen.

Det hele er inspirert av biologi og avstanden til de nærmeste naboene. Det er tatt hensyn til at gjeteren ikke kan se lengre enn en gitt lengde, samt at den søker den nærmeste sauen. Når den nærmeste sauen er lokalisert, er det den som er i fokus, og algoritmen deres tar utelukkende hensyn til en sau av gangen. På grunn av sauenes flokk-oppførsel vil den nærmeste sauen bytte ofte og man får en gjete-oppførsel. Slik kjøres gjetingen uten sentralisert styring.



Figur 6: Gjeting av droner fra et område til et annet [14].

Studien fra 2017 [14] har fokus på både gjeting av sauene og samling. Denne studien har, som den fra 2022, også sett på gjeting langs en linje og ikke en bane med vinkler. I tillegg har denne studien sett på en algoritme gjeterne følger, noe som kan ligne på en tilstandsmaskin, uten at det er direkte nevnt. Gjeterne er i forskjellige tilstander; "samling", "gjeting", "hold på samme sted". Denne måten å tenke på gjeting med tilstander er relevant for denne oppgaven, samtidig er selve gjetingen også relevant.

3.4 Oppsummering

Sett i sammenheng med andre temaer for masteroppgaver finnes det lite tidligere relevant arbeid i forbindelse med dronebasert sanking av sau. Det arbeidet som finnes har en matematisk tilnæring i kontrast med denne oppgaven som har en programvarebasert tilnærming. Denne spesifikke oppgaven om dronebasert sanking av sau har NTNU laget ny dette året (2023). Det finnes derfor ikke identiske masteroppgaver fra tidligere år ved NTNU. Det tidligere arbeidet gjennomgått i dette kapitlet er oppsummert i tabell 1 under.

Beskrivelse	**Forfatter**	**År**
Behavior-Based Herding Algorithm for Social Force Model Based Sheep Herd [8]	He Cai, Yaqi He, Jinye Wu, Huanli Goa	2023
Herd guidance by multiple sheepdog agents with repulsive force [15]	Masao Kubo, Midori Tashiro, Hiroshi Sato, Akihiro Yamaguchi	2022
Autonomous Shepherding Behaviors of Multiple Target Steering Robots [14]	Lee, Wonki and Kim, DaeEun	2017

Tabell 1: Oversikt over tidligere arbeid.

4 Metode

I dette kapittelet er det beskrevet hvordan de forskjellige delene i oppgaven er løst og hvordan de er satt sammen til en komplett programvare. Det er beskrevet hvilken teknologi som er brukt og hvordan programflyten er satt opp for å få kontrollalgoritmen og visualiseringen til å kjøre. Simuleringen av sauene og hvordan stiene i utmarken er satt sammen er forklart på et høyt nivå, mens kontrollalgoritmen er utdypet i detalj. Dette i samsvar med fokus i oppgaven.

4.1 Teknologi

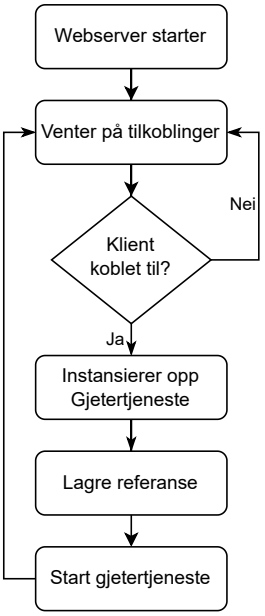
Kontrollalgoritmen er skrevet i programmeringsspråket C# [7] med hjelp av plattformen .NET [1]. I tillegg er web rammeverket ASP.NET [3] tatt i bruk for å enkelt interagere med et brukergrensesnitt som er skrevet i HTML, javascript og CSS. Teknologien gir et godt grunnlag for å utvikle kontrollalgoritmen, hvor programmeringsspråket har statiske typer og plattformen har god støtte for matematikk, deriblant operasjoner på vektorer. HTML og javascript gjør det enkelt å tegne streker og linjer med enkle operasjoner. HTML har et innebygget element med navn "canvas" som er laget for tegning [9]. Fra javascript kalles det metoder for å tegne på kanvaset, for eksempel sirkler som representerer sauer. CSS er tatt i bruk til å plassere kanvas og knapper på rett plass i HTML siden. For å sende data mellom kontrollalgoritmen og visualiseringen er det brukt et bibliotek, SignalR [18]. SignalR bruker websockets som tillater kontrollalgoritmen å sende en kontinuerlig strøm av data til brukergrensesnittet. Denne dataen er i hovedsak koordinater til hvor kanvaset skal tegne linjer eller sirkler. Ut over disse teknologiene er det, med unntak av tilstandsmaskinen, ikke brukt noen rammeverk eller pakker. Dette er et bevist valg som er tatt på bakgrunn av at man ønsker å ha kontroll på så store deler av systemet som mulig. Ved å ha denne kontrollen er det mulig å introdusere litt og litt kompleksitet i alle deler av systemet. Dette tillater å løse oppgaven iterativt og raskt forkaste en iterasjon om den ikke fungerer.

4.2 Programflyt

Med god hjelp av teknologien nevnt i forrige kapittel er det bygget opp en programflyt med fokus på frakobling av kontrollalgoritmen fra det som har med simuleringen å gjøre.

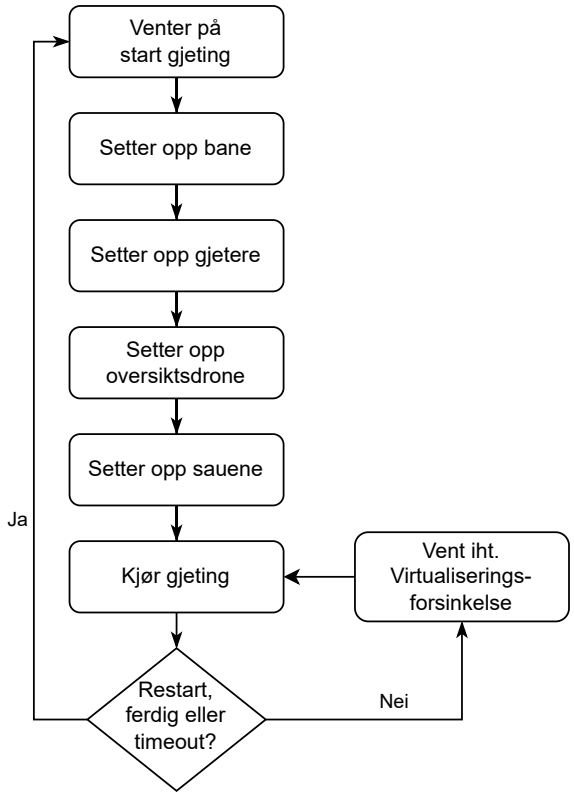
Figur 7 visualiserer programflyten når applikasjonen starter opp. Med ASP.NET følger det med en webserver med navn Kestrel [13]. Denne webserveren starter opp og tilgjengeliggjør applikasjonen. Webserveren kan håndtere mange tilkoblinger i parallell med å lytte etter nye. Lyttingen kjøres i en egen oppgave og for hver klient som kobler seg til, opprettes det en ny oppgave. Når en klient kobler seg til, instansierers det opp en gjetertjeneste som kjøres i en egen oppgave. En referanse til denne tjenesten settes inn i en trådsikker oppslagsbar liste, med oppslagsnøkkel lik klientens tilkoblings-id. Når en klient gjør en endring i brukergrensesnittet brukes tilkoblings-id til å slå opp

i listen og hente ut rett gjetertjeneste til å gjøre endringen på. Når referansen er lagret i listen, startes gjetertjenesten.



Figur 7: Flytskjema for webserver.

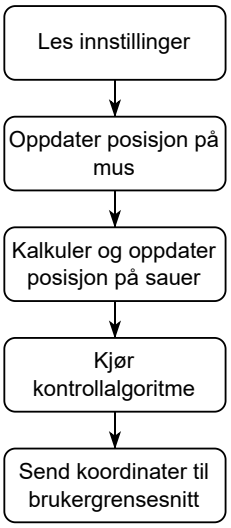
Gjetertjenestens programflyt er visualisert i figur 8. Først venter den på startsignal fra brukergrensesnittet. Når startsignal er gitt, settes banen som er valgt opp. Deretter instansieres gjeterne, oversiktsdronen og sauene med oppstartskoordinater. Når dette er gjort, starter selve gjetingen. Gjetingen kjøres kontinuerlig til restart fra brukergrensesnittet er forespurt eller at alle sauene er inne i målposisjon, altså at gjetingen er ferdig. I tillegg er det laget funksjonalitet som avbryter gjetingen etter en gitt tid. Dette er i hovedsak brukt for å avbryte automatiske tester når man antar at noe har gått galt i gjetingen.



Figur 8: Flytskjema iht. gjetertjeneste.

Stegene som inngår i selve gjetingen er visualisert i figur 9. Først leses det inn innstillingene for gjetingen. Innstillingene er "virtualiseringsforsinkelse" og "hvilkårlighetsfaktor" til sauene. Gjetingen har ingen kobling mot eller forhold til tid. Dette gjør at man med innstillingen "virtualiseringsforsinkelse" kan bestemme hvor fort gjetingen skal kjøre. Maks hastighet er så fort datamaskinen klarer å eksekvere all koden, uavhengig av om brukergrensesnittet klarer å henge med. Når man ønsker å se hva som foregår kan man legge inn en forsinkelse mellom hver eksekvering, via innstillingen "virtualiseringsforsinkelse". Ved testing gjennom brukergrensesnittet er det hensiktsmessig å kjøre visualiseringen så sakte at man får med seg hva som foregår. Ved automatisk testing uten et brukergrensesnitt, er det hensiktsmessig å kjøre så fort som mulig, for å slippe å vente på at testene skal fullføre.

Når innstillingene er lest inn oppdateres en manuell gjeter sin posisjon. Gjeteren kontrolleres av brukerens musepeker i brukergrensesnittet, derfor oppdateres musen sin posisjon i koden, slik at sauene kan påvirkes av den. Når dette er gjort, kalkuleres det og settes nye posisjoner til sauene. Når sauene har fått sin nye posisjon, er det klart for å finne ut hvordan dronene skal bevege seg. Dette skjer i kontrollalgoritmen, som er hovedfokus i denne oppgaven. Dronenes posisjoner settes, deretter sendes det et sett med koordinater til brukergrensesnittet, slik at alt kan tegnes opp. Etter dette sørger gjetertjenesten fra figur 8 for at gjetingen kjøres om igjen etter gitt forsikelse i henhold til "visualiseringshastighet".



Figur 9: Flytskjema for gjeting.

I parallell med disse programflytene forklart over, lytter webserveren kontinuerlig på kommandoer/innstillinger fra brukergrensesnittet. Når en kommando mottas, brukes klientens tilkoblings-id til å slå opp i listen med gjetertjenester for å hente ut tjenesten som hører til klienten. Deretter oppdateres tjenesten med kommandoen. Når èn eksekvering er ferdig leses alle kommandoer og innstillinger inn. Finnes det en endringen i en kommando eller innstilling vil neste eksekvering utføres i henhold til den. Er det for eksempel sendt stopp-kommando vil gjetingen stoppes.

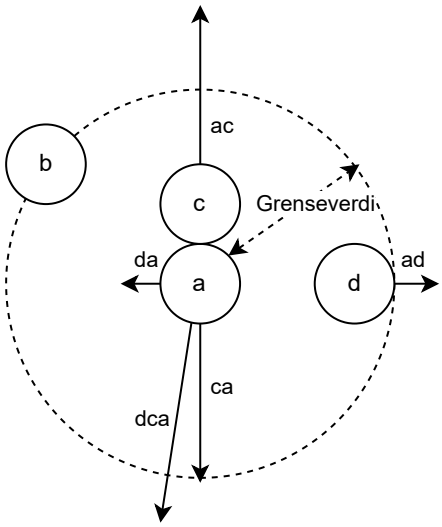
4.3 Simulering av sauer

For å kunne teste kontrollalgoritmen er det simulert en flokk med fem sauer. Simuleringen baserer seg på krefter som virker mellom sauene og gjeternes påvirkning på sauene. I programvaren er disse kreftene kalkulert med vektorregning. Hver kraft har en vekting slik at for eksempel å bevege seg bort fra dronene får større kraft, enn å holde sammen som en flokk. De forskjellige kreftene summeres opp og resulterer i en vektor til punktet sauen skal befinne seg i neste eksekvering av programmet. Metoden er inspirert av artiklene fra tidligere arbeid i kapittel 3 [15][8][14] og basert på kontinuerlig testing.

Den første kraften som virker på sauen er dens naboer. Naboer er sauer som er i umiddelbar nærheten av sauen. Denne kraften er i hovedsak tatt med for å unngå at sauer går over hverandre. Figur 10 visualiserer nabokreftene. Sirklene med bokstaver i representerer sauer, hvor a er sauen kraften regnes ut for. For at kraften skal påvirke sauene, må naboen være innenfor en grenseverdi, dette er den stiplede sirkelen. Denne grenseverdien er lav, da det er normalt at sauer står ved siden av hverandre, men ikke over hverandre. Kreftene som virker er de svarte pilene. Som figuren viser, påvirkes ikke sau a av sau b, siden den ikke er innenfor grenseverdien. Sau c og d påvirker sau a med forskjellig kraft, summen av denne kraften er påvirkningen på sau a, merket med dca.

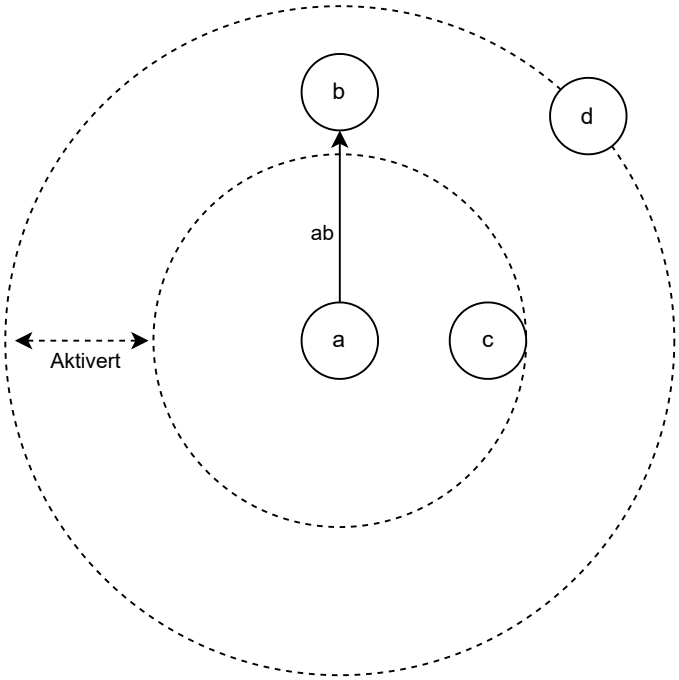
Siden denne kraften har som oppgave å ikke plassere sauer over hverandre, virker den eksponentielt kraftigere jo nærmere en sau kommer en annen.

For hver eksekvering av koden, kalkuleres kraften for alle sauene før de flytter seg. Fra eksempelet i figur 10 betyr dette at nabokraften som virker på sau c vil være tilnærmet lik som for sau a, men litt annen på grunn av sau d sin posisjon og sau b. Sau d vil bli påvirket av sau a og c, mens sau b vil bli påvirket av sau c.



Figur 10: Kraft som gjør at sauene ikke går over hverandre med utp. i sau a.

Den andre kraften som virker inn på sauene er en kraft laget for at de skal holde sammen som en flokk. Denne har på samme måte som nabokraften, grenseverdi for når den virker, men slutter å virke når sauene er nære hverandre. Dette gjør at den har et felt hvor den er aktiv, da den også slutter å virke når sauene er for langt borte fra hverandre. Dette simulerer at flokken har splittet seg i to eller flere flokker. Figur 11 visualiserer kraften med utgangspunkt i sau a. Sau d og c befinner seg i områder hvor kraften ikke påvirker sau a. Sau b påvirker sau a slik at den trekker mot sau b iht. vektor ab. Siden kraften blir regnet ut for alle sauene vil det skape en flokk oppførsel. For eksempel vil sau d bli trukket mot sau b og c, mens sau b mot a og d. På samme måte som nabokraften, virker denne også eksponensielt, men har andre parameter enn nabokraften. For å forklare hvordan en flokk kan deles opp i to flokker, legges det frem følgende eksempel: Man har en flokk med fire sauer. Denne kan deles opp i to flokker om for eksempel en sau befinner seg utenfor den ytterste sirkelen fra figur 11 til alle de tre andre sauene. Det kan også bli delt opp i to flokker viss for eksempel to sauer befinner seg utenfor de to andre sine ytterste sirkler.

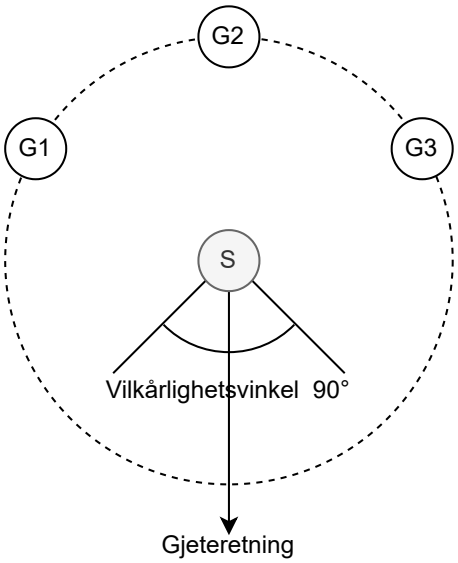


Figur 11: Kraft som gjør at sauene holder sammen som en flokk med utp. i sau a.

En tredje kraft er implementer for å bevege sauene bort fra gjeterne. Denne er implementert på samme måte som nabokraften, men med en høyere grenseverdi slik at sauene presses bort tidligere. I tillegg har denne kraften en høyere vekting enn både nabokraften og kraften som holder flokken sammen. Dette betyr at flokken kan brytes opp i to flokker om gjeterne gjeter på en slik måte at de presser en eller flere sauer for langt fra hverandre.

I utmarken finnes det stier og stikryss. For å simulere at sauene ikke alltid ønsker å ta samme retning i et stikryss som gjeterne, er det implementert en kraft som slåes på når flokken nærmer seg et kryss. Dette er en fast kraft med en innjustert vekting basert på testing. Denne kraften gjør at sauene får en naturlig tiltrekning mot venstre utgang, i henhold til avgrensningene i kapitel 2.3.

Sauer er dyr, de vil derfor ha en viss grad vilkårlighet når de beveger seg. For å få med denne kompleksiteten er resultatet av kreftene nevnt tidligere i dette kapittelet summert som en vektor, og denne vektoren er rotert før de sendes til sauene. Hvor mye de er rotert kommer ann på hva som er satt i innstillingen "Vilkårlighetsfaktor" kalt f, men begrenset til $1 => f <= 100$. Vinkelen kalkuleres slik: $\frac{\pi}{100} \cdot f = v$, hvor f er vilkårlighetsfaktoren. Denne begrensningen er satt fordi det er naturlig å anta at sauene vil holde noe av retningen mens de gjetes. De vil mest sannsynlig ikke snu seg å løpe rett mot dronene. Vilkårlighetsvinkelen er visualisert i et eksempel i figur 12. I figuren representerer $G1-3$ gjeterdronene og S en sau i saueflokken. Sauen gjetes nedover i figuren slik pilen viser. Vilkårlighetsvinkelen i eksempelet er $90°$, dette resulterer i at sauen kan vilkårlig bytte retning $\pm 45°$.



Figur 12: Vilkårlighetsvinkel som gjør at sauene har en grad av vilkårlig retning under gjeting.

For å forsikre at sauens vilkårlige bevegelse er identisk for hver gjeting, sendes det inn et nummer kalt "vilkårlighetsfrø" som C# sin "Random" metode tar som argument. Dette gjør at om gjetingen kjøres flere ganger, vil sauene ha en form for vilkårlig bevegelse, men den vilkårlige bevegelsen er lik for hver gjeting, så lenge vilkårlighetsfrøet er det samme. Dette gjør at man kan reprodusere kjøringen av en feilet gjeting, for å finne feilen og utbedre den.

For å oppsummere hvilke krefter som virker inn og når de virker inn på sauene, er de listet opp i tabell 2.

Beskrivelse	**Aktivert**	**Deaktivert**	**Virkning**	**Vekting**
Intimesone	< 15	$15 <$	Frastøt	0.3
Flokkdannelse	$100 < x < 200$	< 100 eller $200 <$	Tiltrekking	0.1
Gjeting	< 100	$100 <$	Frastøt	1.0
Stikryss	< 100	$100 <$	Tiltrekking	1.1

Tabell 2: Oversikt over parameter for sauesimulering.

4.4 Simulering av gjetere

I forbindelse med den komplette simuleringen av gjeting, er gjeterne simulert for å få bevegelse på sauene. Som nevnt i kapittel 4.3 vil gjetere som er innen for en grenseverdi ha en frastøtende effekt på sauene.

Gjeteren mottar et koordinat fra kontrollalgoritmen som er tiltenkt at kjører på oversiktsdronen. Simuleringen i gjeterdronen kalkulerer en vektor mellom nå-posisjon og koordinatet den har fått

beskjed om å forflytte seg til. Denne vektoren normaliseres først og deretter multipliseres den med en kostant som er gjeterens hastighet. På denne måten får gjeteren konstant fart mot koordinatet den har fått beskjed om å forflytte seg til.

I tillegg til gjeterne som styres av oversiktsdronene, er det laget en ekstra gjeter som kan styres av musepekeren i brukergrensesnittet. Denne er brukt under manuell testing for å teste ulike scenarier det er vanskelig å fremprovosere med de andre gjeterne.

4.5 Kontrollalgoritme

I dette kapittelet beskrives den sentrale delen i oppgaven, kontrollalgoritmen. For å strukturere opp programvaren er følgene hjelpemoduler bygget og brukt av algoritmen:

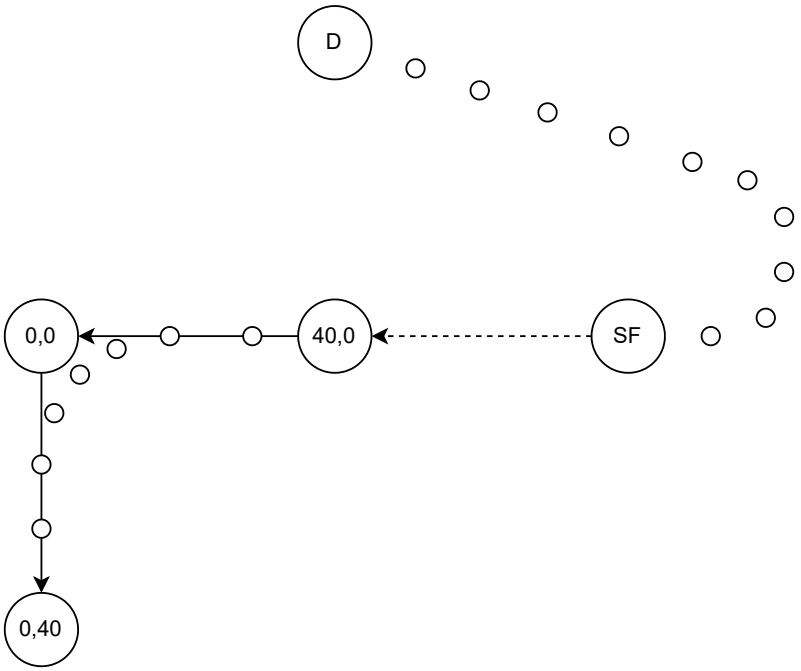
- Banekoordinator
- Punktkalkulator
- Generell kalkulator
- Tilstandsmaskin

Disse modulene, samt programflyten og grensesnittet til algoritmen, er forklart nærmere i de påfølgende underkapitlene.

Figur 13 er tatt med for å spesifisere hvilke moduler som gjør hva. I figuren er D dronene, SF saueflokken, og koordinatene (40,0), (0,0), (0,40) er koordinater på banen saueflokken skal gjetes etter. Figuren tar utgangspunkt i når gjetingen starter. Banekoordinatoren holder kontroll på koordinatene, kvittere ut de man har vært på, og stikryss. Punktkalkulatoren brukes til å regne ut de små sirklene på figuren, kalt punkter. Dette er punktene dronene flyr etter for å få beveget saueflokken i riktig retning. Det vises to scenarier på figuren. Det første scenariet er vist på figuren med sirklene fra dronene frem til saueflokken. Det forekommer når dronene kommer fra retningen banen går mot, da må dronene kjøre rundt på baksiden av saueflokken. Det andre scenariet er vist på figuren med sirklene fra koordinat (40,0), gjennom (0,0) og avslutter i (0,40). Da brukes punktkalkulatoren til å bygge opp en rundet sving, for å få en mer naturlig gjeting og unngå at sauene går rett frem. Er svingen for slak følges koordinatene direkte uten avrundet sving.

Den generelle kalkulatoren inneholder flere grunnleggende funksjoner, som for eksempel regne ut vinkelen på en sving i banen.

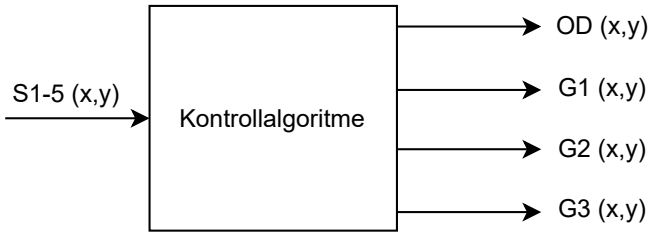
Administreringen av når disse modulene brukes har en overordnede styringslogikk som tilstandsmaskinen står for.



Figur 13: Oppgaven til kontrollalgoritmens hjelpemoduler.

4.5.1 Grensesnitt

Kontrollalgoritmen har ingen koblinger mot modulene som har med simulering å gjøre. Den er satt opp slik at den skal kunne tas ut og implementeres i et virkelig system. Det er derfor satt opp et grensesnitt mot algoritmen som representerer slik det er naturlig å implementere den. Dette grensesnittet er representert i figur 14. En liste med de fem sauenens posisjoner sendes inn i algoritmen og ut kommer fire kommandoer, en til oversiktsdronen og en til hver av gjeterdronene.



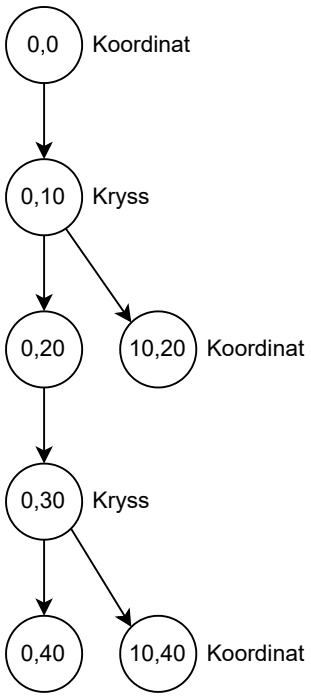
Figur 14: Grensesnitt kontrollalgoritme.

4.5.2 Banekoordinator

Banekoordinatoren er en egen modul og har en sentral rolle i kontrollalgoritmen. Den har til en hver tid informasjon om hvor stiene i terrenget er og eventuelle stikryss. I tillegg har den informasjon om hvor på banen gjeterne har vært før og lar de kvittere ut punkter de er på.

Banekoordinatoren holder en linket liste av koordinater og stikryss som tillater utstikkere av listen.

Listen er visualisert i figur 15. Sirklene er punkter på gjeteområdet, med koordinat i (x,y) format. Hver punkt på banen er av typen koordinat eller kryss. Stikryssene er forenklet til å ha en inngang og to utganger, i tillegg er det begrenset til at sauene alltid vil ta venstre utgang, mens dronene vil gjete sauene til høyre utgang.



Figur 15: Banekoordinator - Datastruktur.

Banekoordinatoren har flere oppgaver og tilbyr følgende grensesnitt:

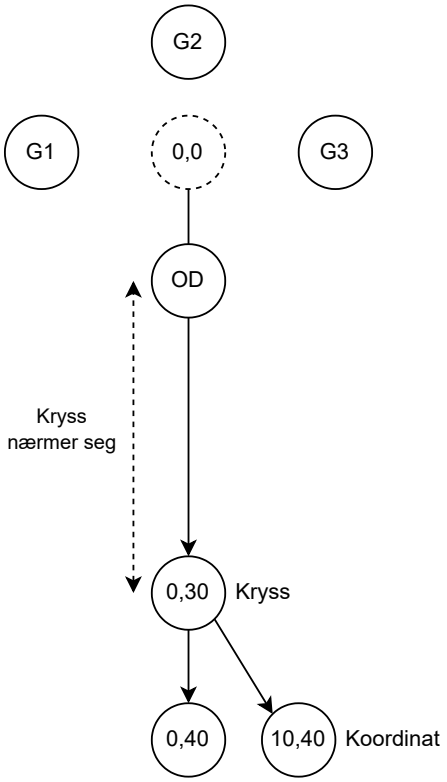
- Hent gjeldende koordinat.
- Hent neste koordinat.
- Nærmer et kryss seg?
- Hent neste kryss.
- Regn ut nærmeste koordinat.
- Kvitter ut punkt.
- Returner hele banen.
- Returner resterende bane for droner eller sauer.

Styrelogikken i oversiktsdronene er satt opp til å følge punkter langs stien. For å vite hvilke punkt den gjeter mot, hentes dette fra banekoordinatorens grensesnitt "Hent gjeldende koordinat". For å vite hvor dronen har vært, kvitterer den ut punktene langs stien, slik at den drives mot neste

punkt når den når et punkt. Det er banekoordinatoren som holder kontroll på punktene og hvilke som er kvittert ut. Grensesnitt for å hente ut gjeldende og neste koordinat brukes til å finne ut om dronene må svinge for å komme til neste koordinat, og hvor mye de skal svinge.

Om neste kryss nærmer seg, brukes til å kontrollere om en gjeter skal sendes frem og sperre utgang i kryss. Scenarioet er visualisert i figur 16. $G1-3$ representerer gjeterdronene. OD er oversiktsdronen.

Banekoordinatoren tilbyr i grensesnittet å gi beskjed til kontrollalgoritmen om et stikryss nærmer seg, gitt ut i fra en grenseverdi. Det vil si at om stikrysset er nærmere en den gitte grenseverdien vil banekoordinatoren returnere "ja" på metoden "Nærmer et kryss seg?". Metoden "Hent neste kryss" brukes i kalkulasjonen hvor en gjeter skal sendes ut og sperre utgangen i krysset, for å posisjonere gjeteren i henhold til krysset.

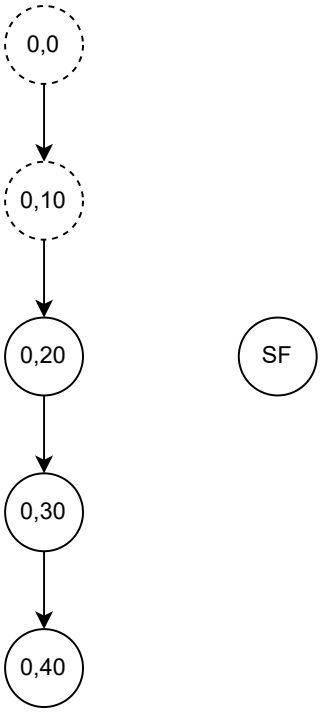


Figur 16: Banekoordinator - Detektering av "stikryss nærmer seg".

Etter hvert som gjetingen beveges fremover sørger kontrollalgoritmen for å kalle opp banekoordinatoren for å kvittere ut koordinater. Dette gjøres slik at kontrollalgoritmen kan forholde seg til få koordinater av gangen.

Banekoordinatoren tilbyr også "Regn ut nærmeste koordinat" som brukes når flokken skal gjetes fra utmark til bane. Når saueflokken ikke står på enden av en bane, er det hensiktsmessig å gjete flokken til nærmeste punktet, i steden for å gjete de helt tilbake til banens start. Dette er visualisert i figur 17. Punkt SF representerer saueflokken. Som man ser, er det hensiktsmessig å gjete flokken til punkt (0,20), i steden for (0,0), da målposisjon er (0,40). Banekoordinatoren tar seg av å kvittere

ut (0,0) og (0,10) slik at kontrollalgoritmen tar utgangspunkt i at (0,20) er første punkt på banen.



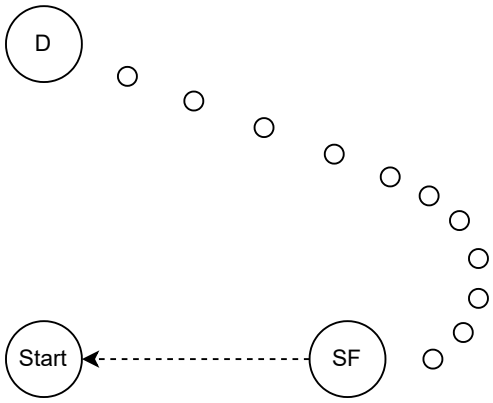
Figur 17: Banekoordinator - Kvittering til- og utregning av nærmeste punkt.

I tillegg har banekoordinatoren metoder som brukes for visualisering i brukergrensesnittet. "Returner hele banen" brukes for å få alle koordinatene til banen slik at den kan tegnes opp i brukergrensesnittet. "Returner resterende bane for droner eller sauer" tar inn om man ønsker bane for droner eller sauer og returnerer den resterende banen. Denne brukes for å kunne ha en annen farge på resterende bane i forhold til hele banen i brukergrensesnittet.

4.5.3 Punktkalkulator

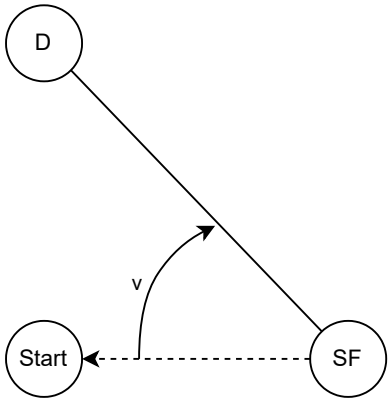
Punktkalkulatoren har i oppgave å regne ut punkter mellom et koordinat og et annet for å lage en naturlig bane til dronene. Den har to hovedoppgaver; beregne punkter når flokken hentes og beregne punkter ved gjeting når vinkelen på banen er under $\frac{\pi}{3}$. De to tilfellene er visualisert i innledningen på figur 13 med små sirkler. Første tilfelle til høyre og andre tilfelle til venstre.

Beregning av punkter ved henting av flokk: Når gjetingen starter er første oppgave å fly ut til saueflokken og gjete de til nærmeste punkt på banen bonden har definert. Om man skal gjete flokken i motsatt retning av den dronene kommer fra, trenger man å komme på baksiden av flokken, slik at de ikke drives i feil retning. Denne utfordringen er visualisert i figur 18. D er dronene, SF er saueflokken og $Start$ er første punktet på banen sauene skal gjetes til. Punktene dronene følger er de små sirklene, det er disse Punktkalkulatoren regner ut. Som vist på figuren, flyr dronene i en omvei for å komme på baksiden av flokken når de skal gjetes mot $Start$.



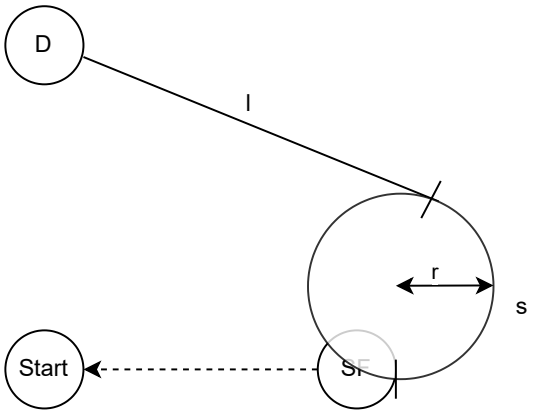
Figur 18: Punktkalkulator - Utfordringen den løser for henting av flokk.

Det første som gjøres er å kalkulere vinkelen mellom dronenes startposisjon, saueflokken og første punkt sauene skal gjetes til. Dette er visualisert i figur 19, hvor v er vinkelen som kalkuleres. Er vinkel $v < 150°$, slik som på figuren, settes det opp punkter for å fly til baksiden av flokken, mens er vinkel $v >= 150°$, vil dronene kunne fly rett til saueflokkens posisjon.



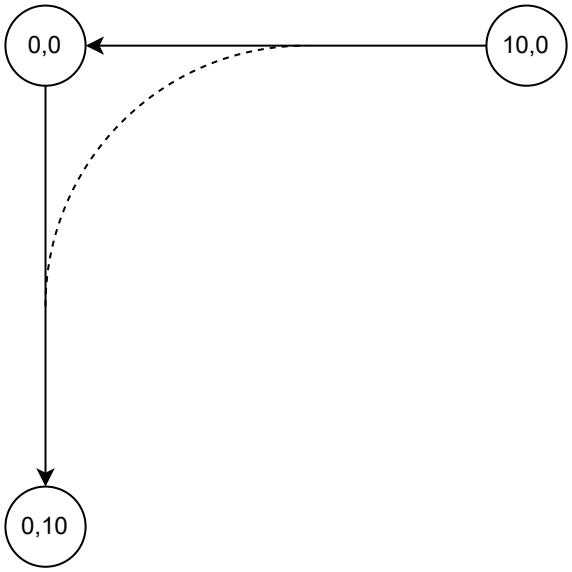
Figur 19: Punktkalkulator - Kalkulering av vinkel.

For å bygge opp en bane, tas det utgangspunkt i en rett linje og en sirkel med fast radius. Dette er visualisert i figur 20. Den rette linjen er markert med l og sirkelen med fast radius r. Hvor mye som brukes av sirkelen, det vil si omkretslinjen s kalkuleres ut i fra v fra forrige figur. Det brukes en fast verdi, pluss en linær faktor av v. Deretter settes startpunktet til l i dronenes midtposisjon og sluttpunktet til l settes i startpunktet til s. Punktene kalkuleres langs l ved en løkke som kalkulerer hvert punkt med fast avstand mellom de, hvor endringen i x-retningen og y-retningen fordeles på disse punktene. Punktene langs s kalkuleres ved en løkke som fordeler lengden av s ut over punktene med fast avstand mellom de. Dette gjøres ved å finne hvor senter av sirkelen må være, deretter kalkulere punktene ved hjelp av cosinus og sinus multiplisert med radiusen og justert for utgangsposisjon. Resultatet av beregningene er en liste av punkter som vist i figur 18.



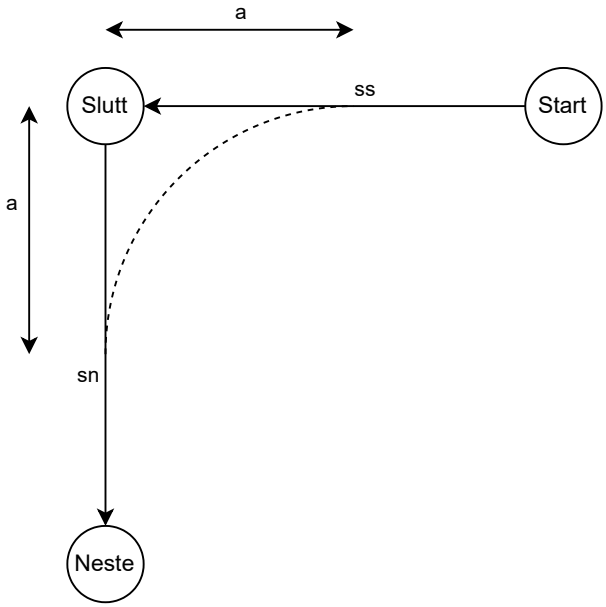
Figur 20: Punktkalkulator - Kalkulering av punkter.

Beregning av punkter ved gjeting langs sti: Ved gjeting langs sti kan det forekomme vinkler på stien hvor det er viktig at dronene holder en naturlig vinkel på sauene for å unngå å gjete de av stien. Dette gjøres ved å avrunde vinklene slik at dronene forbereder sauene på at det skal svinge i samsvar med stien. I figur 21 er utfordringen punktkalkulatoren løser visualisert. Stien satt opp av bonden er de svarte linjene via koordinatene (10,0), (0,0), (0,10). Stien inneholder en vinkel på 90°. Denne vinkelen ønsker man å runde av slik at ikke sauene presses rett frem og av stien. Den ønskede flybanen til dronene er visualisert med den stiplede linjen i figuren.

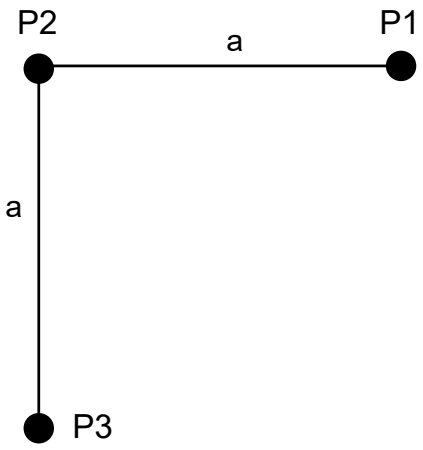


Figur 21: Punktkalkulator - Utfordringen den løser for vinkler.

Figur 22 presenterer første steg i utregningen. Først regnes startpunktet og slutt punktet til kurven ut. Dette gjøres ved å trekke av en konstant a fra vektoren ss og sn. I tillegg brukes punktet "Slutt". Med disse tre punktene har man utgangspunktet til å kalkulere en Bézier kurve [4], vist i figur 23.



Figur 22: Punktkalkulator - Kalkulering av start og slutt på kurve.



Figur 23: Punktkalkulator - Utgangspunkt til kalkulasjon av bézierkurve.

Funksjonen for kalkuleringen av kurven er hentet fra en artikkel [5] som har implementert den i Python. Den er oversatt til C# og følger kode 1.

```
1 private double P(double P0, double P1, double P2, double t)
2 {
3     return Math.Pow(1 - t, 2) * P0 + 2 * t * (1 - t) * P1 + Math.Pow(t, 2) * P2;
4 }
```

Kode 1: Implementasjon av Bézier kurve i C#.

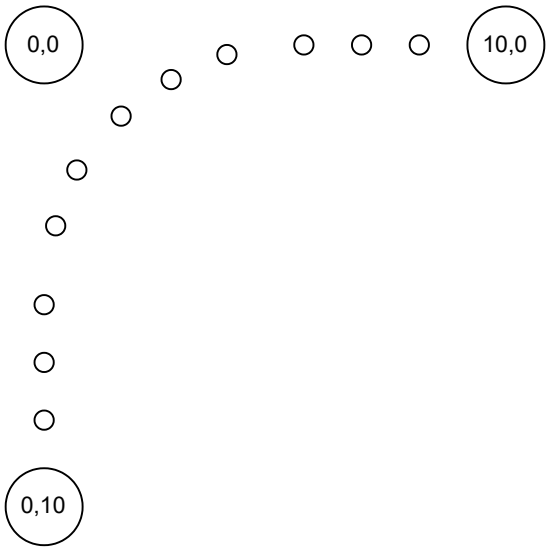
Deretter er koordinatene kalkulert i kode 2 ved bruk av funksjonen i kode 1. Dette gjøres ved å kalkulere punkter fra $t = 0$ til $t = 1$ med steglengde 0.1. Implementasjonen av løkken er på linje

4. På linje 6 kalkuleres x-verdien og på linje 7 kalkuleres y-verdien. Når dette er gjort lages det et kvitterbart koordinat. Dette kvitterbare koordinatet legges inn i en liste som dronene flyr etter.

```
1  public List<AckableCoordinate> Bezier(Coordinate startCurve, Coordinate
   ↪  zeroPoint, Coordinate endCurve, int indexStart)
2  {
3      var list = new List<AckableCoordinate>();
4      for (var t = 0.0; t <= 1.0; t = t + 0.1)
5      {
6          var x = P(startCurve.X, zeroPoint.X, endCurve.X, t);
7          var y = P(startCurve.Y, zeroPoint.Y, endCurve.Y, t);
8          list.Add(new AckableCoordinate(indexStart++, x, y, true));
9      }
10     return list;
11 }
```

Kode 2: Kalkulasjon av punkter på Bézier kurve.

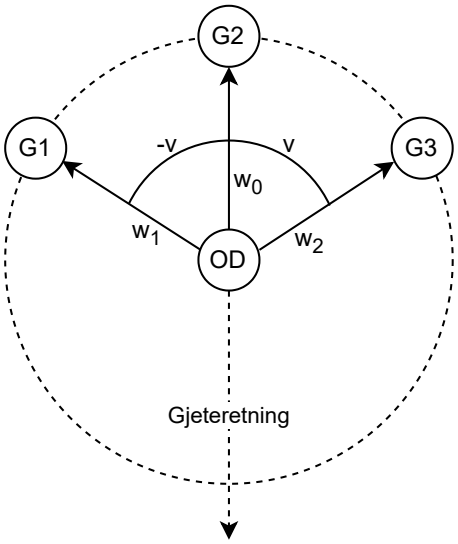
Når punktene i kurven er kalkulert, settes det inn punkter for de rette linjene før og etter kurven. Dette gjøres ved å kjøre en løkke langs resterende lengder av vektorene *ss* og *sn*. Ved å spesifisere indeksene i objektet "KvitterbartKoordinat" kan punktene sorteres på indeksen og man oppnår en liste av punkter i henhold til figur 24. Da har dronene punkter å fly etter som representerer en naturlig bane for sauene, slik at det er mindre sannsynlighet for at sauene beveger seg ut av stien.



Figur 24: Punktkalkulator - Resultat for vinkel.

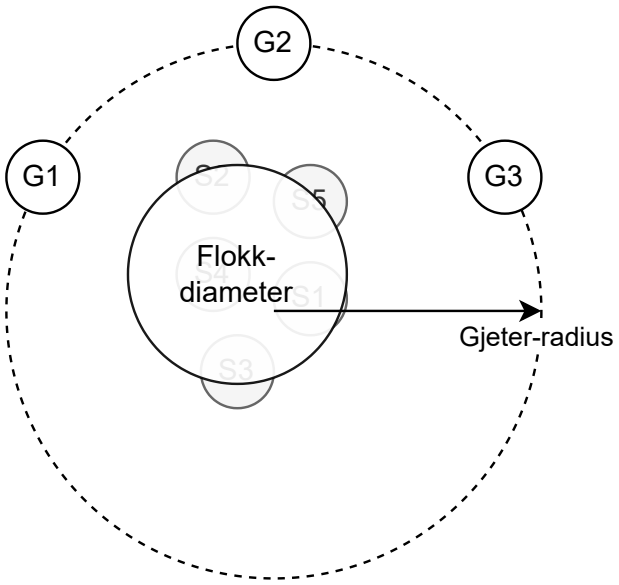
4.5.4 Posisjonering av gjetere

Gjeterene er dronene som flyr for å skremme sauene i riktig retning. Posisjonen til gjeterne tar utgangspunkt i å ligge på en sirkel rundt oversiktsdronen. Figur 25 visualiserer posisjoneringen. Her er gjeterne $G1$, $G2$, $G3$ og oversiktsdronen OD. Først plasseres $G2$ ved å kalkulere vektor w_0 ved å legge på en gitt lengde fra oversiktsdronens posisjon i motsatt retning av koordinatet det gjetes mot. Deretter legges vinkel v til w_0 og man får w_2. Ved å trekke vinkel v fra w_0 får man w_1. Vektorene w_{0-2} legges til oversiktsdronenes nåværende posisjon og man får gjeternes posisjon ved normaltilstand.

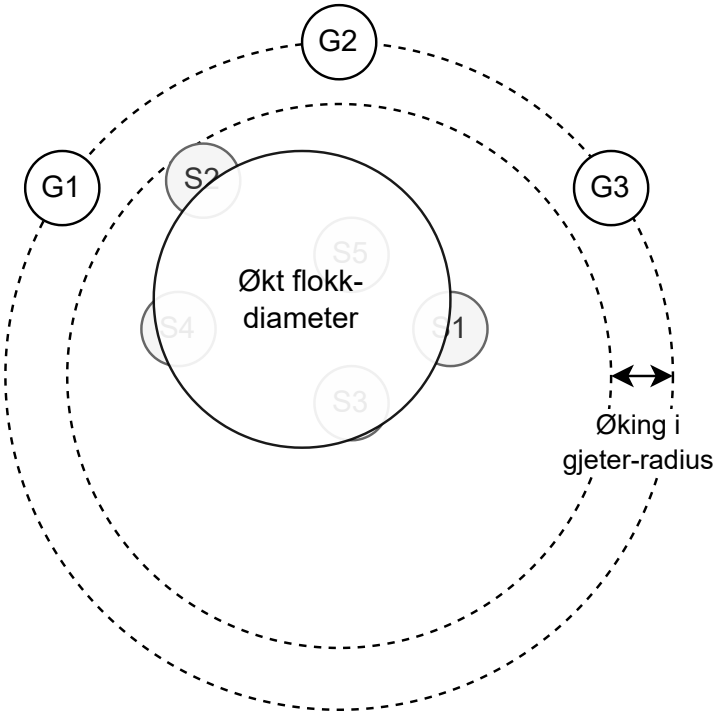


Figur 25: Posisjonering av gjetere - Standardposisjon.

Saueflokken holder ikke alltid like tett sammen. Som et tiltak mot å miste sauer fra flokken, justeres lengden til w_0 fra figur 25, altså radiusen til sirkelen gjeterne ligger på. Dette er visualisert i figur 26, hvor gjeterne er $G1-3$ og sauene er $S1-5$. Figuren viser lav gjeteradius i (a) øverst og hvordan radiusen øker er vist i (b) nederst. Radiusen er visualisert med den delvis gjennomsiktige sirkelen man ser sauene gjennom. Ny gjeter-radius kalkuleres ut i fra saueflokkens diameter ved følgende formel: $(\frac{diameter}{2})^{1.2} = gjeterradius$. I tillegg er det satt minimums- og maksimumsgrense for hvor stor gjeter-radiusen kan være.



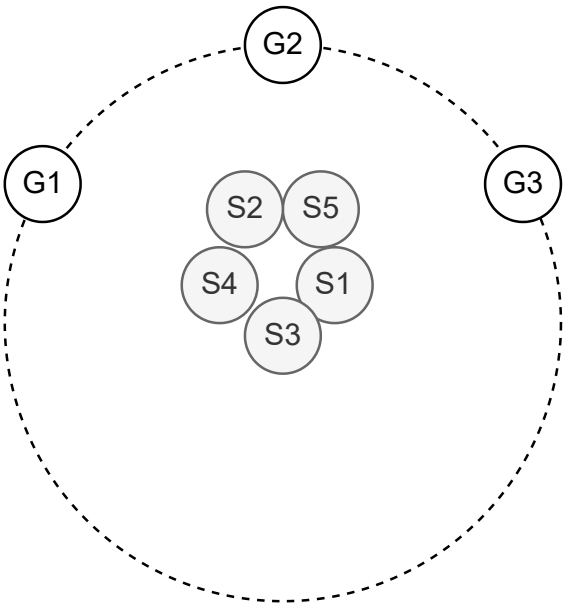
(a) Normal gjeter-radius.



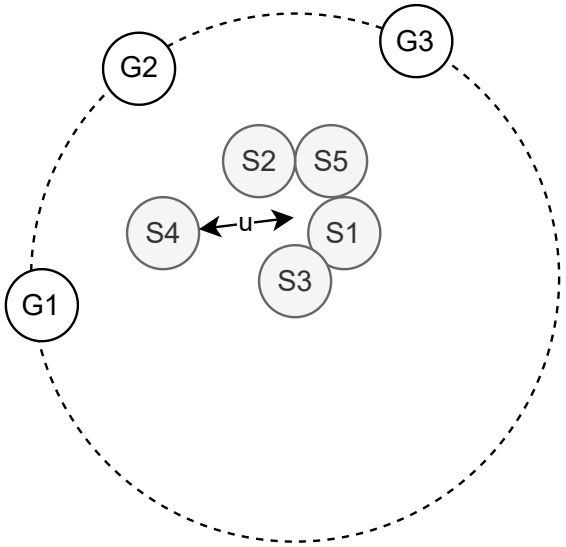
(b) Økt gjeter-radius.

Figur 26: Posisjonering av gjetere - Økt gjeter-radius.

Selv om sauene blir gjetet på en sti, er det tatt nøyde for at enkelte sauer kan prøve å stikke av. Figur 27 visualiserer dette hvor $S4$ er på vei ut av flokken i (b). Som et tiltak for å forhindre dette, regnes det kontinuerlig ut en vektor u fra saueflokkens senter, ut til ytterste sau. Retningen på vektor u brukes til å justere vinkelen gjeterne har mot oversiktsdronen. Som figuren viser, brukes vektor u til å vinkle gjeterne slik at de møter $S4$ på dens vei ut av flokken.



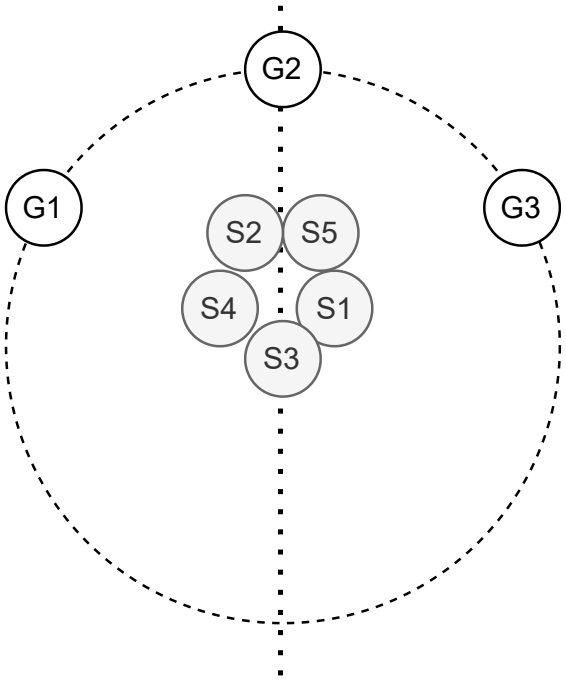
(a) Normal posisjon.



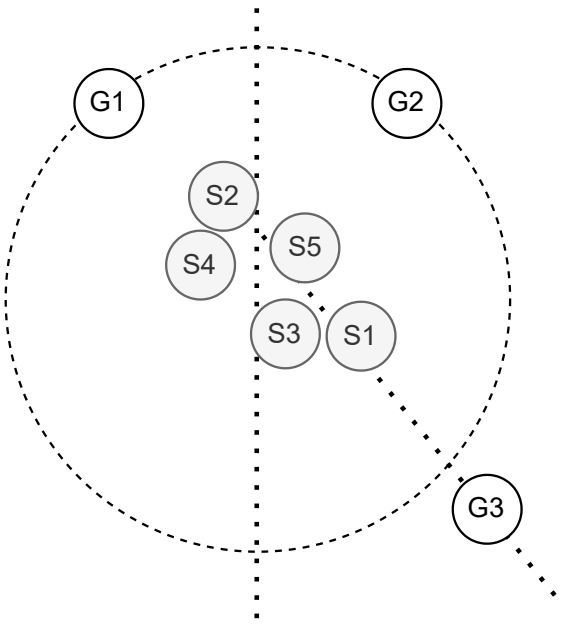
(b) Rotert posisjon.

Figur 27: Posisjonering av gjetere - Rotasjon for utspringer.

Ute i naturen gjetes sauene oftes på en sti. Stier finnes det mange av og man vil møte stikryss. Ved stikryss er det sannsynlig at sauene ønsker å bruke en annen utgang en hvor de skal gjetes. Dette tas høyde for ved en stikryss-manøver. Manøveren er visualisert i figur 28, hvor gjeterene og sauene er som på de forrige figurene i dette kapittelet. Øverst (a) i figuren gjetes sauene mot et stikryss. Nede (b) er stikryss-manøveren visualisert. Her ser man at gjeter $G3$ er sendt i forveien for å blokkere utgangen sauene ikke skal benytte. Samtidig er $G1 - 2$ sine posisjoner rotert slik at de har best mulig kontroll på flokken. I figur 29 viser kalkulasjonen som blir gjort. På et tidspunkt forteller banekoordinatoren at et stikryss nærmer seg. Vektor a sin retning kalkuleres fra koordinatet midt i krysset og neste punkt på stien (0,20). Lengen til vektor a er en konstant verdi.

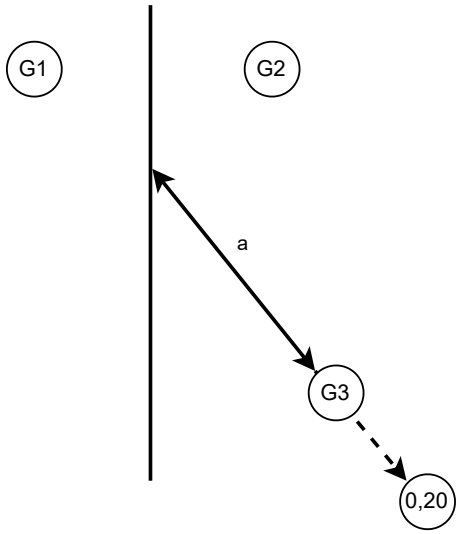


(a) Posisjon før stikryss.



(b) Posisjon i stikryss.

Figur 28: Posisjonering av gjetere - Stikryss.



Figur 29: Posisjonering av gjetere - Kalkulasjon av posisjoner ved stikryss.

4.5.5 Tilstandsmaskin

For å forenkle logikken det er behov for i kontrollalgoritmen, er det tatt i bruk en tilstandsmaskin med åpen kildekode fra Github [10]. Tilstandsmaskinen tilbyr et grensesnitt for å opprette tilstander, sette opp hvilke tilstander som kan gå til hvilke, konfigurere opp betingelser for å bytte tilstand, samt eksekvering av kode når man entrer en tilstand. De forskjellige tilstandene og betingelsene for å oppnå de er visualisert i figur 30 og 31. De overordnede tilstandene er i figur 30 mens undertilstandene til "Hent flokk" og "Følg bane til koordinat" er i figur 31.

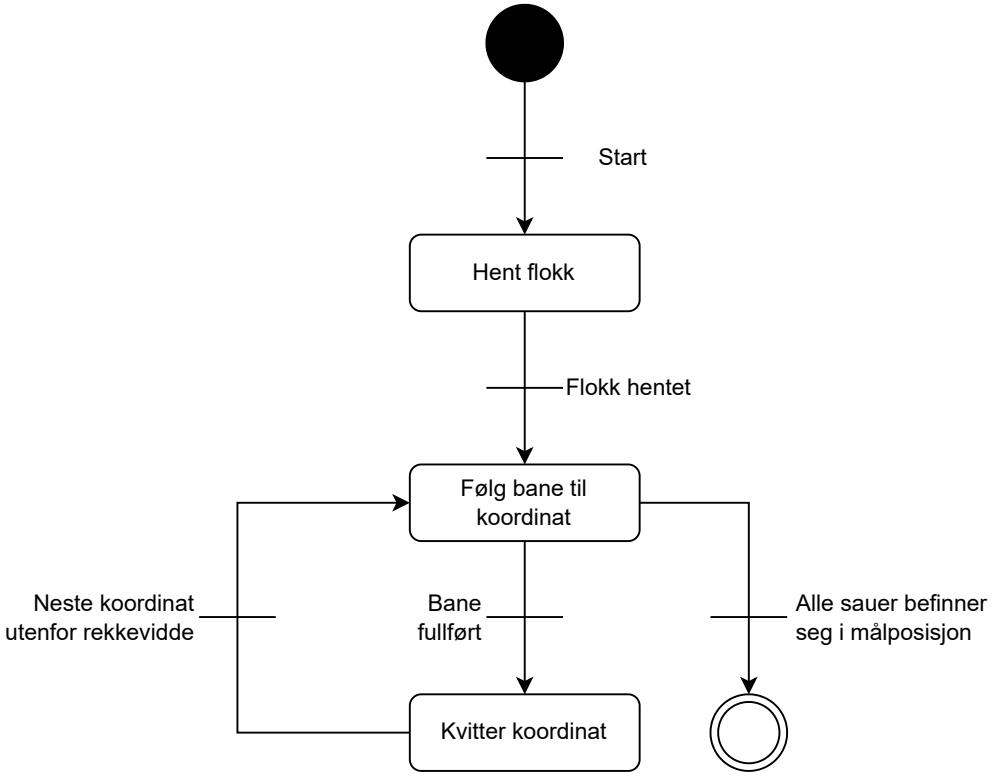
Gjetingen starter ved startsignal. Startsignalet settes fra brukergrensesnittet. Da entrer tilstandsmaskinen "Hent flokk". Ved entring av denne tilstanden utføres flere operasjoner. Først kalkuleres saueflokkens senter, dette gjøres for å vite hvilket koordinat dronen skal forholde seg til når de flyr ut mot flokken. Deretter brukes banekoordinatoren til å kvittere ut de punktene som ligger lenger fra målposisjonen enn saueflokken er, dette er i henhold til figur 17 i kapittel 4.5.2. Da vet algoritmen hvor sauene er og hvilket punkt de skal til først.

Avhengig av vinkelen mellom punktet sauene skal gjetes til, punktet saueflokken står på og dronenes posisjon, kalkuleres det ett sett med punkter ved hjelp av punktkalkulatoren fra kapittel 4.5.3. Deretter følges punktene ved hjelp av undertilstandene vist i figur 31. Ett punkt kvitteres ut ved at oversiktsdronen befinner seg på punktets posisjon, ved en gitt feilmargin. Når alle punktene på vei til saueflokken er kvittert ut, oppnås kriteriet "Flokk hentet" og tilstanden går over til "Følg bane til koordinat". Ved entring av "Følg bane til koordinat" brukes punktkalkulatorer for å lage et nytt sett med punkter fra dronens posisjon frem til første koordinat på stien. Deretter følges samme logikk som når flokken ble hentet.

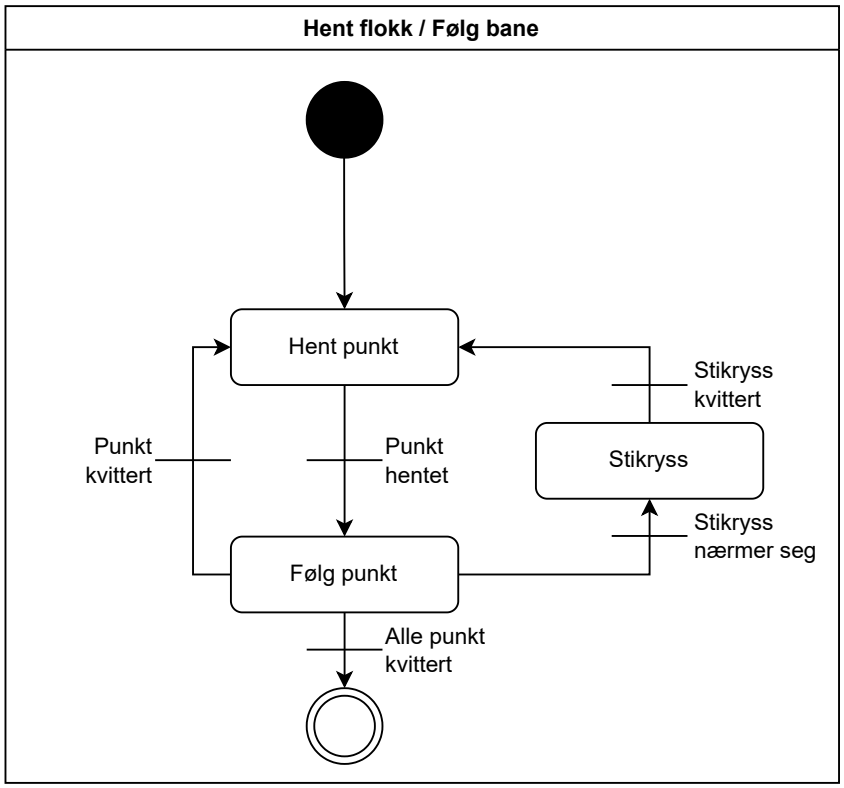
Undertilstandene har også en tredje tilstand, "Stikryss". Det er banekoordinatoren som gir beskjed

til tilstandsmaskinen om det nærmer seg et stikryss. Denne tilstandens brukes til å kontrollere når en gjeter skal sendes ut for å blokkere i et stikryss. Dette ble nærmere forklart i kapittel 4.5.4.

Når alle punktene fra punktkalkulatoren er kvittert ut, kvitteres dette koordinatet som utført i tilstanden "Kvitter koordinat". Dette gjøres i banekoordinatoren. Ved kvittering henter banekoordinatoren frem neste koordinat som skal nås og setter det til "nåværende koordinat". Koordinatet er da utenfor rekkevidde, som gjør at tilstanden går over i "Følg bane til koordinat" igjen. Slik skifter tilstandene frem til alle sauene befinner seg i målposisjonen.



Figur 30: Tilstandsmaskin - Overordnede tilstander.



Figur 31: Tilstandsmaskin - Underordnede tilstander.

4.5.6 Generell kalkulering

For å strukturere koden samtidig som det gir bedre testbarhet, er det laget flere statiske klasser som tilbyr generelle kalkuleringer brukt flere plasser i algoritmen. I dette kapitlet listes noen av disse kalkuleringene opp, det forklares hvordan de gjøres og eksempeler på bruk.

Vinkel mellom to vektorer: Bruker $atan2$ til å kalkulere vinkelen mellom to vektorer. Brukes for eksempel når vinkelen mellom dronene, saueflokken og første punkt på banen skal kalkuleres.

Senter av flokk: En flokk inneholder en liste av sauer, som igjen inneholder koordinatet sauer befinner seg. For å finne senter av flokken summeres x-verdiene og y-verdiene hver for seg. Deretter deles disse to summene på antallet sauer i flokken, altså elementer i listen. Dette gir et nytt koordinat som er senter av flokken. Funksjonen brukes for eksempel til å sette et koordinat på hvor flokken befinner seg for å kunne kalkulere om det finnes en utspringer.

Eksponentiell frastøt: Et sett med matematiske funksjoner for å etterligne sauens bevegelse når for eksempel gjeteren nærmer seg. Når gjeteren nærmer seg minker vektoren mellom gjeteren og sauen, dess nærmere gjeteren kommer jo fortere springer sauen bort. Hoved-metoden returnerer en vektor som gir sauen kraft til å springe bort.

Rotere vektor: Matematisk funksjon som bruker *cos* og *sin* til å rotere en vektor. Brukes for

eksempel til å posisjonere gjeterne på sidene, etter at vektoren til gjeteren i midten er kalkulert.

Innen rekkevidde: Matematisk funksjon som sjekker om et koordinat er innen en rekkevidde av et annet koordinat. Denne er også laget med hysterese. Brukes for eksempel til å trigge kvittering av punkt på bane.

Utspringer: Metode for å kalkulere hvilken sau som er sauen lengst fra senter av flokken, kalt utspringer. Metoden tar inn en liste med koordinater, et senterkoordinat og returnerer koordinatet til utspringeren. For å kalkulere utspringere, går en løkke gjennom alle koordinatene i listen og kalkulerer avstand til senter. Den med lengst avstand returneres. Denne metoden brukes når gjeternes posisjon skal korrigeres for utspringer, nærmere forklart i kapitel 4.5.4.

Gruppering av flokker: Metode for å lage en liste med grupper av saueflokker. Metoden tar inn en liste med koordinater og en grenseverdi for hvor stor avstand det kan være til neste koordinat for at to koordinater er i samme gruppe. I hovedsak skal denne metoden returnere bare en gruppe, det vil si at det finnes bare en flokk på 5 sauer. I tilfeller hvor gjetingen feiler og splittes opp i flere flokker, detekterer denne metoden det ved å returnere flere grupper. I tillegg legger metoden til rette for at det i fremtiden kan gjetes flere grupper sekvensielt.

4.5.7 Programflyt

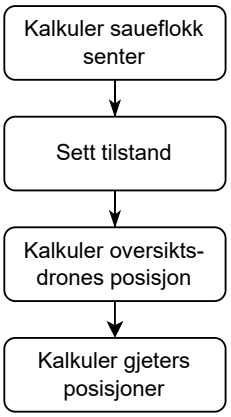
Kontrollalgoritmen kjøres kontinuerlig. Denne kjøringen er det den overordnede programflyten som tar seg av, forklart i kapittel 4.2. I figur 9 finnes det en boks "Kjør kontrollalgoritmen", når denne eksekveres kjøres kontrollalgoritmen. Kontrollalgoritmens sekvens av oppgaver er visualisert i figur 32.

Det første som skjer er at saueflokkenes senter er kalkulert. Siden flokken potensielt kan deles opp i flere flokker, tas det hensyn til. Denne listen kommer ut i sortert rekkefølge med den nærmeste flokken først. Deretter settes den nærmeste flokken til nåværende gjetende flokk. Flokkobjektet inneholder flokkens senter og dens radius til ytterste sau. Dette objektet blir brukt i flere kalkulasjoner senere i programflyten.

Gjetingen kontrolleres av overordnet tilstandsmaskin. Før det gjøres flere kalkulasjoner, oppdateres tilstandsmaskinens tilstand. Dette gjøres ved å sjekke en rekke kriterier og er nærmere forklart i 4.5.5. Når gjetingen går fra en tilstand til en annen kjøres det kode, samtidig som at deler av kontrollogikken baserer seg på hvilken tilstand gjetingen er i.

Oversiktsdronen kontrolleres av hvilken bane som er kalkulert fra punktkalkulatoren eller returnert fra banekoordinatoren. Når tilstanden til gjetingen er satt, settes oversiktsdronens neste posisjon ut til neste punkt på banen som ikke har vært besøkt før.

Til slutt kalkuleres gjeternes posisjoner ut i fra oversiktsdronens. Her tas det hensyn til utspringere og stikryss. Dette ble nærmere forklart i kapittel 4.5.4.



Figur 32: Programflyt kontrollalgoritme.

4.5.8 Gjeting som helhet

De foregående underkapitlene i dette kapitlet har hver for seg forklart hvordan de fungerer og hjelper kontrollalgoritmen. Som helhet, kjører en typisk gjeting følgende sekvens:

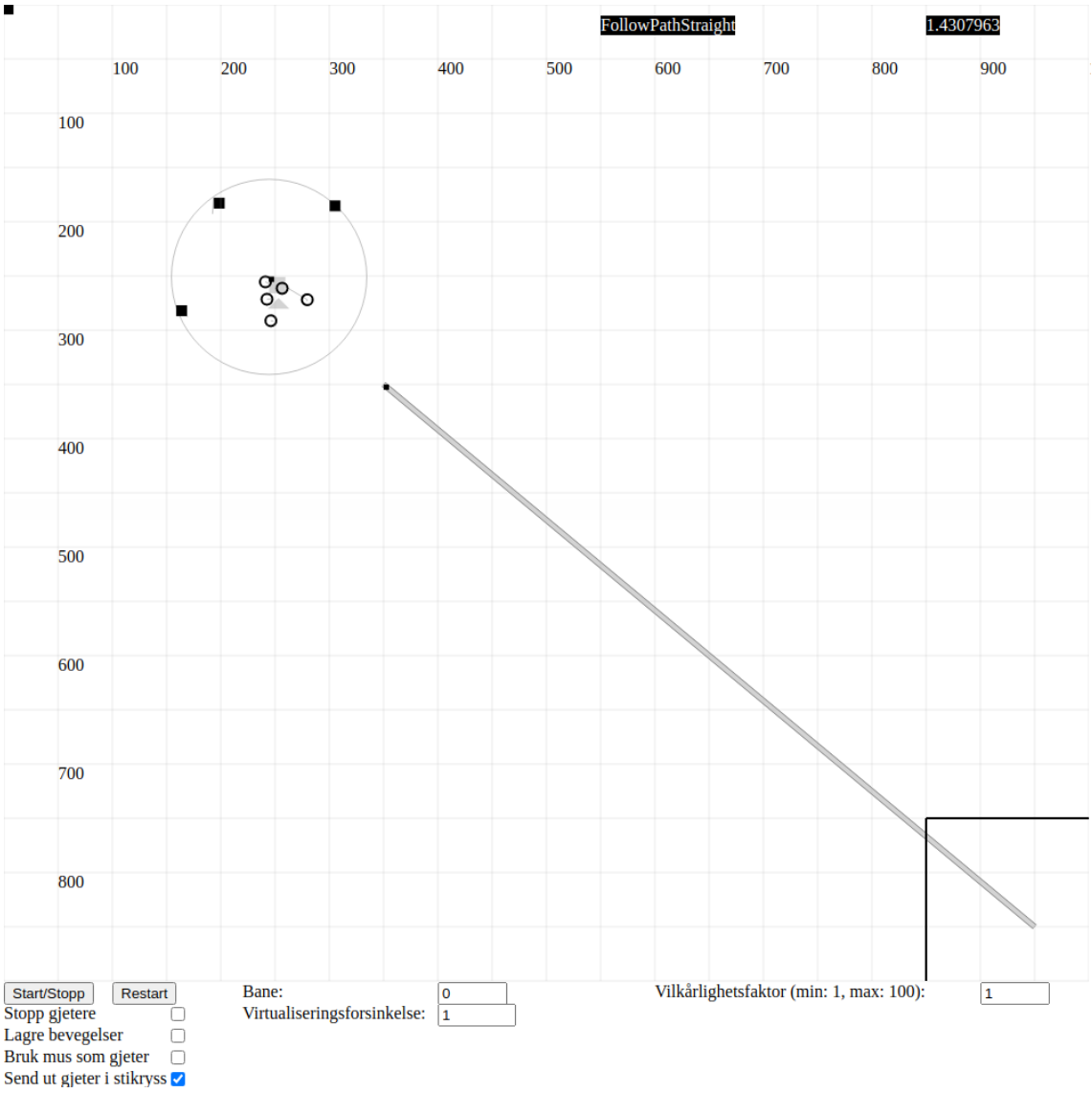
1. Gjetingen startes.
2. Banekoordinatoren kalkulerer det nærmeste punktet til saueflokken.
3. Punktkalkulatoren setter opp punkter til dronene slik at de flyr mot saueflokken og kommer på baksiden av flokken i forhold til det punktet banekoordinatoren har satt til første koordinat.
4. Sauene gjetes fra posisjonen de er på, via alle koordinater på banen og til sluttposisjon, samtidig foregår følgende:
    - Om det er en sving, sørger punktkalkulatoren for en naturlig rundet bane.
    - Om flokken sprer seg, økes avstanden fra gjeterne til senter av flokken.
    - Om en sau prøver å stikke av, svinger gjeterne seg mot sauen.
    - Om det nærmer seg et stikryss, sendes den ene gjeterne ut for å blokkere den ene utgangen.

4.6 Brukergrensesnitt

Brukergrensesnittet er laget enklest mulig for å frigjøre kapasitet til kontrollalgoritmen. Figur 33 viser en skjermdump av grensesnittet. Øverst vises tilstanden systemet er i (FetchingFirstHerd) og tiden siden start (1.1324822) i sekunder. Systemets tilstander er forklart i kapitel 4.5.5. Tallene som går horisontalt og vertikalt er koordinatene for gjeteområdet. Den grå streken er banen sauene skal gjetes på. Sauene befinner seg nå i startposisjon, de er visualisert med en svart ring med hvit fyll.

Gjeterdronene er de svarte firkantene som ligger på den grå sirkelen. Sirkelen er kun en visualisering av hvor gjeterdronene skal fly. Den grå firkanten med en liten svart firkant øverst til venstre er oversiktsdronen. Den grå trekanten med en svart firkant i toppen er saueflokkens kalkulerte senter. Den grå streken fra oversiktsdronene ut til sauen nede til høyre, er visualisering av hvilken sau som er definert som utspringer. Hvordan gjeterne posisjonerer seg i forhold til utspringer er forklart i kapitel 4.5.4. Den svarte firkaten øverst til venstre i figuren er gjeteren som kan styres med musepekeren. På figuren er den deaktivert.

Under gjeteområdet finnes det mulighet for å ta input fra bruker. "Start/Stopp" starter eller stopper hele gjetingen, samme hvor langt den har kommet. "Restart" starter gjetingen på ny. Man kan også huke av om man vil stoppe gjeterene, tegne alle bevegelsene, bruke musen som gjeter og om en gjeter skal sendes mot et stikryss. Til høyre for sjekkboksene kan man velge hvilken bane man vil gjete på og i hvilken fart visualiseringen skal kjøre. Helt til høyre kan man justere i hvilken grad sauenenes retning skal påvirkes av en vilkårlig faktor.



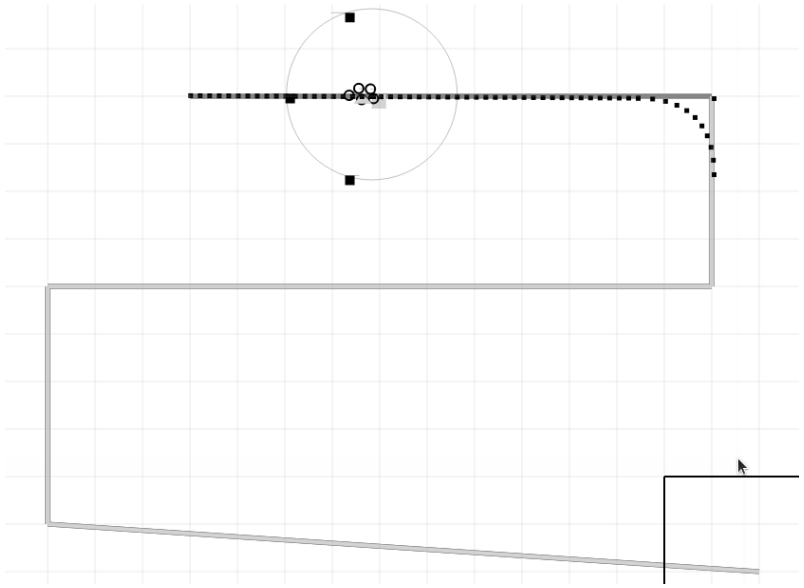
Figur 33: Brukergrensesnitt - Oversikt.

Brukergrensesnittet er til god hjelp under manuell testing og videreutvikling av kontrollalgoritmen. Ved å stoppe gjeterne kan musepeker-gjeteren brukes til å teste simuleringen av sauene i forskjellig scenarier. For å teste funksjonaliteten rundt stikryss er det mulig å velge om en gjeter skal sendes frem og blokkere den ene stien eller om gjetingen skal gå som om det ikke er et kryss. Banen kan velges slik at forskjellige baner kan testes uten å stoppe programvaren. Om man kjører et scenario flere ganger og det blir lenge å vente til gjetingen treffer punktet man tester, kan man justere ned visualiseringsforsinkelsen. Etter hvert som gjetingen går som den skal, er det mulig å justere vilkårlighetsfaktoren slik at vanskelighetsgraden på gjetingen øker.

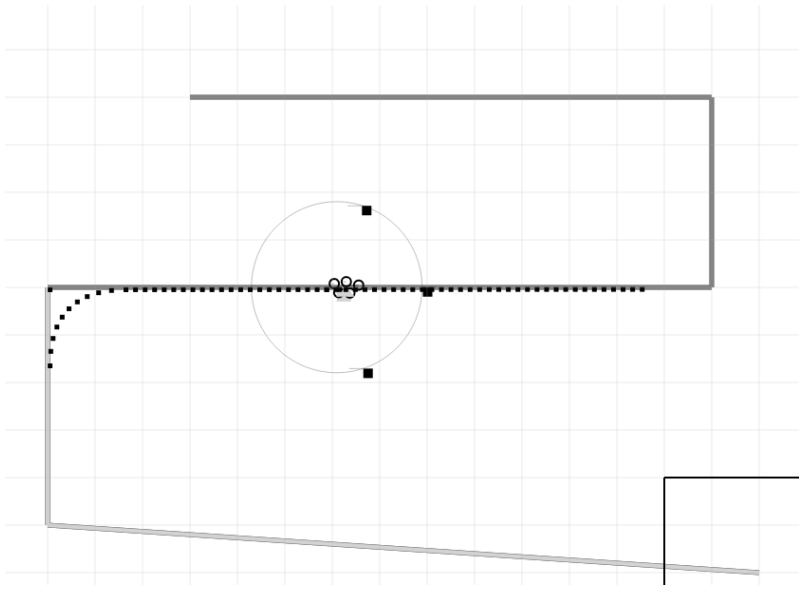
Under gjeting visualiseres det hvor langt på banen man har beveget seg. På figur 34 ser man at det gjetes mot punktet oppe til høyre i figuren. Da er banen man er på mørk grå, mens den banen som gjenstår er lys grå. Når gjetingen kommer lenger bytter fargen, dette vises på figur 35.

Koordinatet det gjetes mot er markert med et lite svart kvadrat. Dette kan man se oppe til høyre i figur 34 og etter hvert som koordinatet oppnås, flyttes kvadratet videre til neste koordinat man vil oppnå.

Fra kontrollalgoritmen er det forklart at det kalkuleres punkter i punktkalkulatoren for å få en naturlig gjetebane. Dette er forklart i kapitel 4.5.3. Disse punktene er visualisert som en rekke av små svarte kvadrater. På figur 34 ser man disse punktene viser at hjørnet oppe til høyre vil rundes av til en naturlig bane.

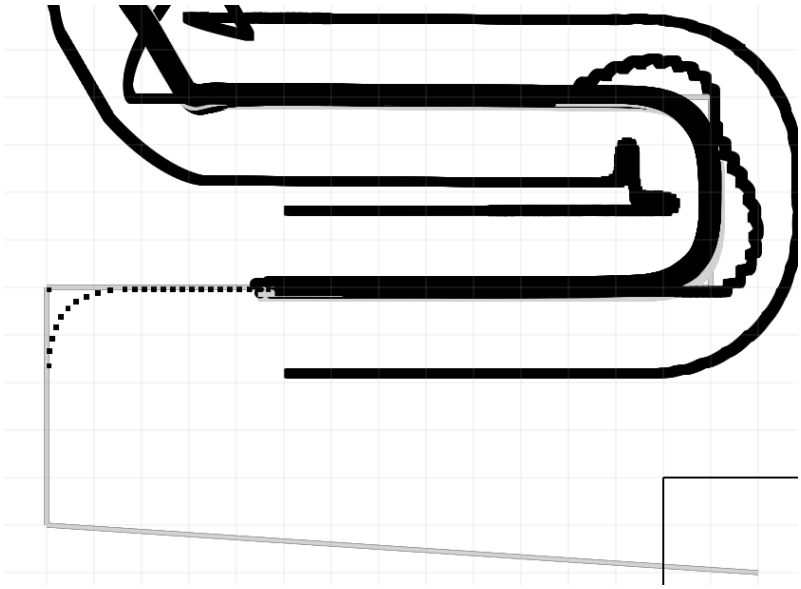


Figur 34: Brukergrensesnitt - Koordinater og punkter mot høyresving.



Figur 35: Brukergrensesnitt - Koordinater og punkter mot venstresving.

For å kunne se tidligere bevegelser og sammenligne to gjetinger, er det laget funksjonalitet for å lagre dronenes og sauenes bevegelser i brukergrensesnittet. I figur 36 er denne funksjonaliteten visualisert i bruk. I figuren ser man at alle bevegelsene er lagret i form av de svarte strekene. Strekene er der hvor dronene har flydd og sauene har løpt. Det er midt i gjetingen som er kommet til ca midt i figuren på vei mot venstre. Om man kjører gjetingen på ny, vil man kunne verifisere at to gjetinger kjører identisk.



Figur 36: Brukergrensesnitt - Lagring av bevegelser.

Oppsummert er brukergrensesnittet utviklet for å teste og visualisere etter hvert som kompleksitet blir introdusert til kontrollalgoritmen.

4.7 Gjeteområde

Området det blir gjetet på er satt til 1000 piksler bredt og 900 høyt, det er det som utgjør rutenettet i Brukergrensesnittet fra kapittel 4.6, figur 33. Området har ikke et forhold til fysiske størrelser. Bevegelsene sauene og dronene gjør er justert inn i forhold til hverandre og gjeteområdet.

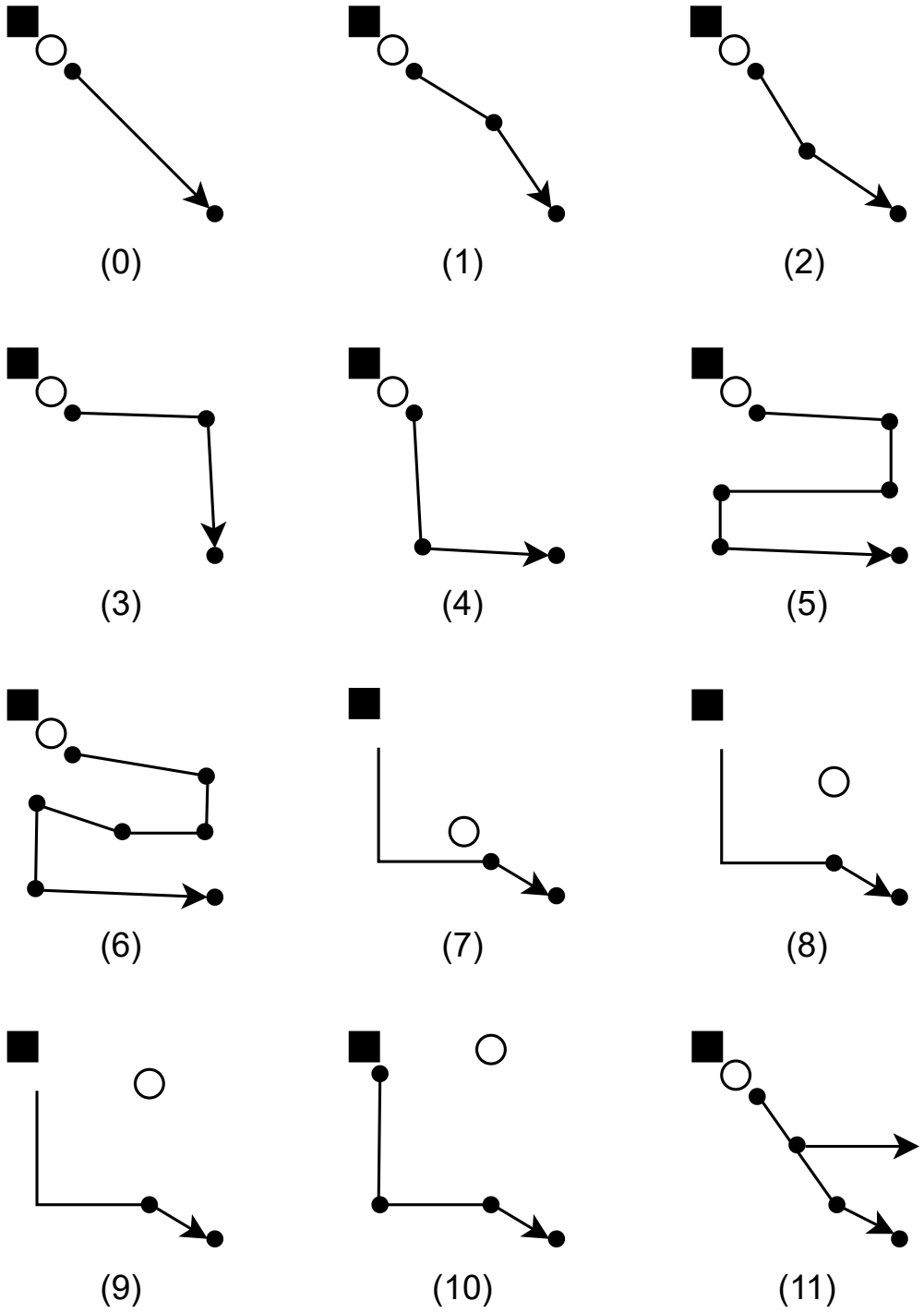
4.8 Testoppsett

For å forsikre at gjetingen fungerer som den skal, er det implementert både enhetstester og ende til ende tester i tillegg til manuell testing. Det er en matematisk tung programvare, denne matematikken er testet med enhetstester. Dette gjelder regning som strekker seg fra koordinatregning, vektorer, vinkler, og til klyngefunksjon.

Ende til ende testene er laget for å teste systemet og gjetingen som helhet. For disse testene er det satt opp baner som gjetingen kjører på. Banene representerer forskjellige scenarier det er ønskelig å

teste. Ved å kjøre disse testene, testes det i tillegg til scenariet også simuleringen og programflyten.

Manuell testing er brukt for å verifisere at systemet fungerer som en helhet og i forbindelse med endringer i funksjonalitet. Det vil si at samspillet mellom visualiseringen, simuleringen, kontrollalgoritmen og programflyten fungerer. Ved endringer i funksjonaliteten er manuell testing brukt for å verifisere at endringen har ønsket effekt.



Figur 37: Testoppsett - Oversikt over baner.

Banene ende til ende testene kjører på, følger figur 37. I figuren er dronene representert med det sorte kvadratet, saueflokken er sirkelen med hvit fyll og koordinatene det skal gjetes via er de små sorte sirklene med målposisjon nede til høyre i hver av figurene. Nummeret under banene i figuren tilsvarer bane nummeret. Man ser for eksempen på bane 9 at sauene skal gjetes fra oppe høyre, til nede høyre, mens i bane 10 skal sauene først gjetes til oppe venstre før banen fører de, via nede venstre, til målposisjon. Banene og scenariene de tester er som følger;

Bane 0: Banen setter opp et scenario hvor kontrollalgoritmen styrer dronene til å gjete sauene fra et punkt til et annet via en rett bane. Dette er den enkleste testen som har det enkleste scenariet. Det er basefunksjonaliteten i gjetingen som testes. Den består av; Banekoordinatorens evne til å holde rede på hvor gjetingen er og kvittere ut koordinater underveis. Oversiktsdronens evne til å kommandere gjeterne til å drive flokke og forhindre utspringere.

Bane 1: Banen bygger videre på bane 0, hvor det i tillegg er lagt til et koordinat som gjør at det skal gjetes gjennom en slak høyresving. Banen verifiserer at det ikke bygges opp punkter for å gjete gjennom svingen, samt at banekoordinatoren håndterer koordinater ut over start og slutt.

Bane 2: Banen har tilnærmet identisk senario som bane 1, men med motsatt sving, det vil si venstre. Samme verifiseringen som for bane 1 gjøres, samtidig forsikres det at ikke noe går galt når svingen er satt opp motsatt.

Bane 3: Banen bygger videre på bane 1, men med en krappere sving som er tilnærmet 90°. Det verifiseres at punktkalkulatoren setter opp en naturlig bane for svingen og at samspillet mellom tilstandsmaskinen, banekoordinatoren og punktkalkulatoren fungerer. Når banekoordinatoren kvitterer ut første koordinat, byttes tilstanden og punktkalkulatoren bes om å kalkulere bane.

Bane 4: Banen har tilnærmet identisk scenario som bane 3, men med motsatt sving, altså venstre. Det verifiseres på samme måte som i bane 3, samtidig som det verifiseres at logikken fungerer for mottsatt sving, det vil si at vinkeler og så videre gjøres riktig.

Bane 5: Banen setter opp et scenario hvor kontrollalgoritmen styrer dronene til å gjete gjennom to u-svinger. Det verifiseres at algoritmen håndterer svinger med lav avstand mellom de, i begge retninger. På banen må samspillet mellom tilstandsmaskinene, banekoordinatoren og punktkalkulatoren jobbe hyppig sammen.

Bane 6: Banen bygger videre på bane 5, men med en krappere sving som er under 90°. Det verifiseres på lik linje som på bane 5, i tillegg verifiseres det at denne krappe svingen håndteres.

Bane 7: Banen setter opp et scenario hvor kontrollalgoritmen må styre dronene til å hente flokken, før den gjetes til målposisjon, via banen. Flokken er på et punkt mellom dronenes startposisjon og første punkt på banen, slik at dronene kan fly i en rett linje for å hente de. Det verifiseres at "hent flokk" funksjonaliteten fungerer og at samspillet mellom tilstandsmaskinen og banekoordinatoren viker som det skal. I tillegg verifiseres det at banekoordinatoren er i stand til å kvittere ut punktene

på banen som er lenger borte fra målposisjon enn hva sauene er.

Bane 8: Banen setter opp et scenario hvor kontrollalgoritmen må styre dronene til å hente flokken på lik linje som bane 7. Forskjellen er vinkelen mellom dronene, saueflokken og første punkt de skal gjetes mot. Vinkelen er helt på grensen til når det bygges opp punkter eller ikke. Det verifiseres at det er hensiktsmessig å bygge opp punkter med denne vinkelen, og at samspillet mellom tilstandsmaskinen, banekoordinatoren og punktkalkulatoren fungerer som den skal.

Bane 9: Banen er satt opp på lik linje som bane 8, men med en krappere vinkel mellom dronene, saueflokken og koordinatet de skal gjetes til. På figuren kommer dronene fra (150,100), saueflokken er på ca (600, 175) og skal gjetes til (600, 600). Flybanen verifiseres i tillegg til samspillet mellom modulene på lik linje som bane 8.

Bane 10: Banen er satt opp på lik linje som 8 og 9, men med en enda krappere vinkel mellom dronene, saueflokken og koordinatet de skal gjetes til. På figuren kommer dronene fra (150, 100), saueflokken er på ca (600, 175) og skal gjetes til (200, 150). Det verifiseres at dronenes flybane ikke påvirker sauene før de er på baksiden av dem. I tillegg verifiseres det også på denne banen samspillet mellom de ulike modulene i kontrollalgoritmen.

Bane 11: Banen er setter opp scenariet hvor sauene skal gjetes gjennom et stikryss. I figuren gjetes saueflokken fra oppe venstre, til nede høyre. De gjetes gjennom stikrysset midt i figuren, hvor sauene trekkes mot utgangen som går til høyre i figuren, mens dronene ønsker å gjete sauene langs banen til målposisjon nede høyre. Det verifiseres at samspillet mellom modulene fungerer, før, under og etter stikryss-manøveren.

For hver bane kjøres det 12 sett med tester. De 12 settene har forskjellig vilkårlighetsfrø, som gjør at bevegelsene til sauene er forskjellig. For hver av disse 12 settene, kjøres det 11 tester med økende vilkårlighetsfaktor, fra 1 til 100 med steglengde på 10. Dette gjør at sauene har høyere grad vilkårlig retning de kan bevege seg i, for hver endring av vilkårlighetsfaktor. Vilkårlighetsfaktoren er oversatt til vinkel i grader i tabell 3. For eksempel, ved en test hvor vilkårlighetsfaktoren er 50, vil sauenes retning bli justert med en vilkårlig vinkel fra $0 - 90°$. Til sammen resulterer dette i $12 \cdot 11 \cdot 12 = 1584$ ende til ende tester. 12 ulike vilkårlighetsfrø, 11 ulike vilkårlighetsvinkler og 12 ulike baner. Vilkårlighetsfaktor og vilkårlighetsvinkel ble nærmere forklart i kapitel 4.3.

Vilkårlighetsfaktor	Vilkårlighetsvinkel
1	$0 - 2°$
10	$0 - 18°$
20	$0 - 36°$
30	$0 - 54°$
40	$0 - 72°$
50	$0 - 90°$
60	$0 - 108°$
70	$0 - 126°$
80	$0 - 144°$
90	$0 - 162°$
100	$0 - 180°$

Tabell 3: Oversettelse fra vilkårlighetsfaktor til vinkel.

Kjøremiljøet til testene er satt opp ved hjelp av "xunit" som er et testrammeverk for .NET. Rammeverket muliggjør enkel kjøring av mange tester og rapportering av resultat. Ende til ende testene er satt opp ved hjelp av "xunit" sin metodikk "Theory". Kodelisting 3 er koden som trengs for testing av èn bane, med èn vilkårlighetsfaktor og ett vilkårlighetsfrø, definert i "InlineData" på linje 2.

Koden i metoden "Herd_Test" kjører èn gang for hver test, konfigurert via "InlineData". Kun en "InlineData" er tatt med for å spare plass. Når selve testen kjører, settes innstillingene, deretter starter gjetingen i "InitializeAndRun()". Denne metoden blokkerer tråden til sauene er i målposisjon eller etter gitt tidsbegrensning. Etter metoden er ferdig eksekvert, sjekkes det om alle sauene er i målposisjonen ("Finished" = true). Er sauene i målposisjon ansees resultatet som positivt, er de ikke, feiler testen.

```
1  [Theory]
2  [InlineData(0, 1, 10)] // Definisjon av testinnstillinger
3  ... (1583 flere)
4  public async Task Herd_Test(int pathNr, int randomAngle, int randomSeed)
5  {
6      var loggerMock = new Mock<ILogger>();
7      var service = new HerdService(loggerMock.Object, null, "testClient",
   ↪  randomSeed, new SheepSettings());
8      try
9      {
10         service.InterceptCross = true;
11         service.PathNr = pathNr;
```

```
12          service.Start = true;
13          service.VisualizationSpeed = 0;
14          service.FailedTimeout = 2.0;
15          service.RandomFactor = randomAngle;
16          await service.InitializeAndRun();
17          service.Finished.Should().BeTrue();
18      }
19      finally
20      {
21          service.Dispose();
22      }
23  }
```

Kode 3: Ende til ende test av gjetertjeneste.

Ved å kjøre testene med flagget "–logger 'html;LogFileName=TestResults.html' " rapporteres resultatene i en HTML fil. Oppsummeringen i rapporten er tatt inn i en Python applikasjon hvor det er skrevet kode for å genererer et søylediagram per bane for å visualisere resultatene.

5 Resultat

I dette kapitlet er resultatene fra oppgaven presentert. Oppgaven har som hovedfokus å utvikle en kontrollalgoritme, med tilhørende forskningspørsmål.

Oppgaven har resultert i en komplett programvare med kontrollalgoritme, visualisering og simulering. Det er mye kompleksitet i en og samme programvare. Det anbefales derfor å teste programvaren gjennom demo på denne linken[1] for å få et helhetlig inntrykk av resultatet. Eventuelt se video av når den kjører på denne linken[2]. Kilekoden ligger åpent på denne linken[3].

Først er kontrollalgoritmens resultater fra ende til ende testing presentert i form av hvilken bane den er testet på og hvordan den presterte på banen. Deretter er resultatene av funksjonaliteten rundt posisjonering av gjeterne presentert ved manuell testing for å vise påvirkningen på gjetingen.

5.1 Ende til ende testing

Kontrollalgoritmens ytelse er testet automatisk med testoppsettet forklart i kapitel 4.8. De automatiske testene består av ende til ende tester og sørger for å kjøre gjeting på forskjellige baner med forskjellige vilkårlighetsfrø og vilkårlighetsvinkel. Gjetingen blir vanskeligere etter hvert som vilkårlighets-vinkelen justeres opp. Vinkelen er justert så langt at tilnærmet alle tester feiler.

Oppsettet resulterer i et sett med resultater per bane. For å få oversikt over scenariet testet i hver av resultatene er det lagt ved en skjermdump. Hva de forskjellige symbolene i skjermdumpen er ble utdypet i kapitel 4.6, brukergrensesnitt.

Det er også lagt ved et søylediagram over hvor mange tester som feiler på de forskjellige vilkårlighetsvinklene. Som nevnt i kapitel 4.8 kjøres det 12 tester på hver bane. Grafen har derfor 12 punkter i y-aksen. For hver bane kjøres det 11 vanskelighetsgrader. Disse er representert med vilkårlighetsvinkelen på x-aksen. Høyere x-verdi betyr høyere vanskelighetsgrad. Høyden på søylen presenterer derfor hvor mange tester som feilet på den gitte banen og vilkårlighetsvinkel. På bane 0, i figur 39 kan man for eksempel se at ved vilkårlighets vinkel 144° har 6 av 12 tester feilet. Det betyr at gjetingen har 50% sjanse for å bli vellykket gjennomført om man antar at sauene kan finne på å løpe 144° i en annen retning en de blir gjetet mot.

For å vise hvilke utfordringer kontrollalgoritmen får i scenariet, er det lagt ved en video for hver bane med en relativt høy vanskelighetsgrad ut i fra resultatet på den gitte banen. En spilleliste med alle videoene er publisert på denne linken[4]. Samme scenario kan kjøres via brukergrensesnitet på demoen nevt i innledningen til dette hovedkapitlet.

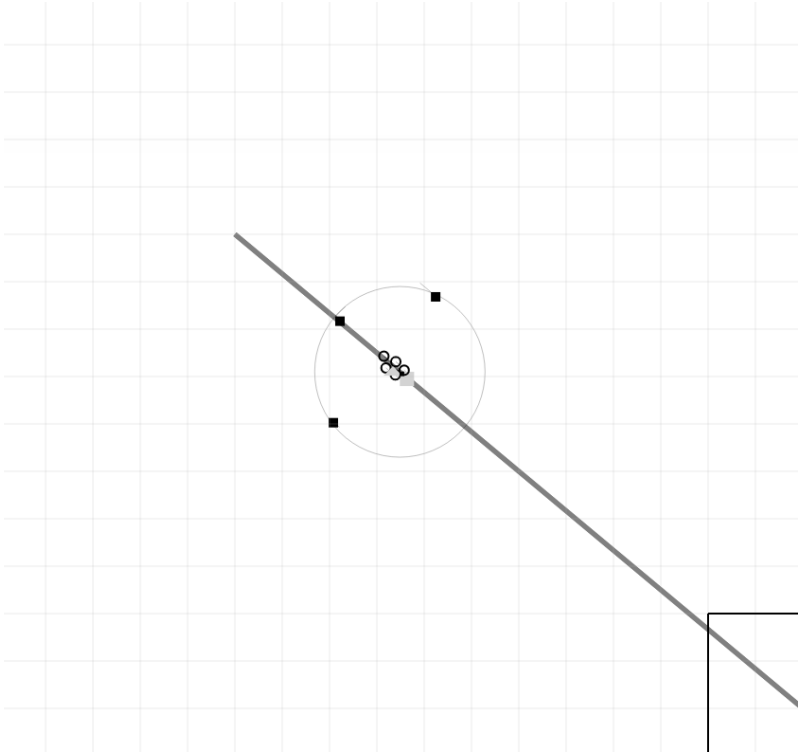
[1] https://sau.vassbo.as

[2] https://bit.ly/gjeting-liste

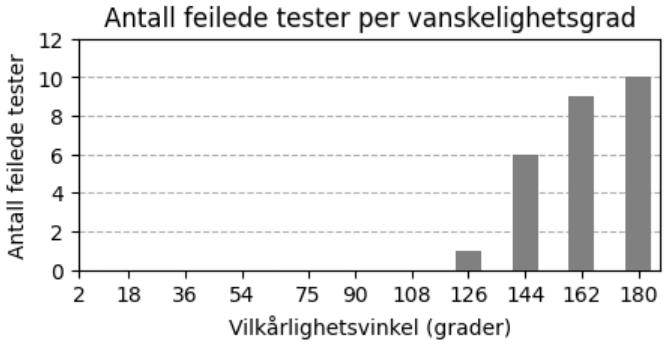
[3] https://github.com/jonev/sheep-herding

[4] https://bit.ly/gjeting-liste

Bane 0 - Rett gjeting: Bane 0 tar for seg et scenario med rett gjeting. Fra skjermdumpen i figur 38 ser man at gjetingen foregår fra oppe venstre til nede høyre langs en rett bane. Fra resultatene i grafen fra figur 39 ser man at det ikke er noen feilende tester før vilkårlighetsvinkelen er oppe på 126°. Ved yttereligere oppjustering av vilkårlighetsvinkelen øker antall feilende tester drastisk, men selv på 180° er det 2 suksessfulle tester. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[5].



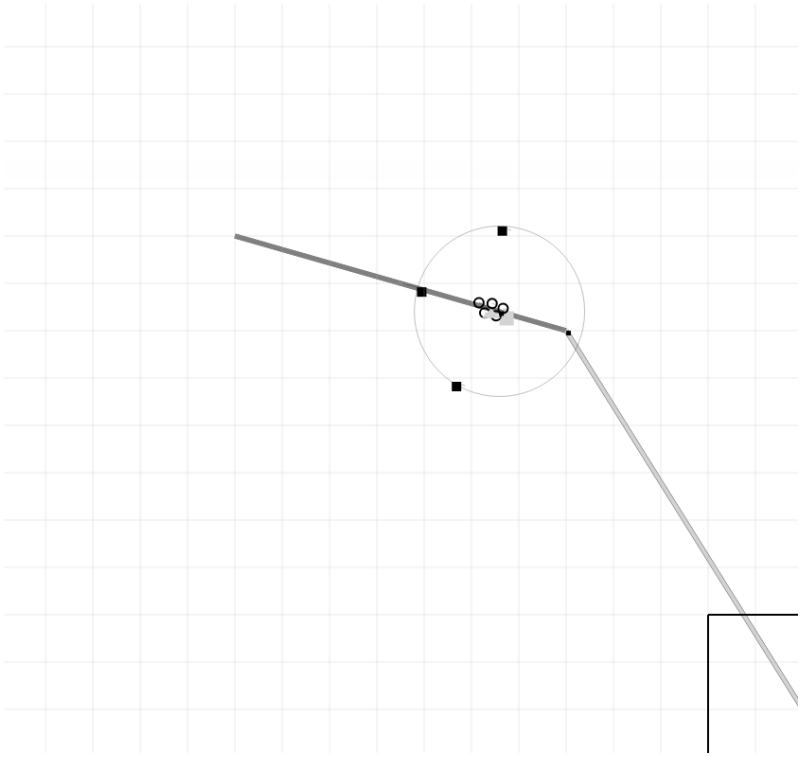
Figur 38: Resultat - Bane 0: Skjermdump rett gjeting.



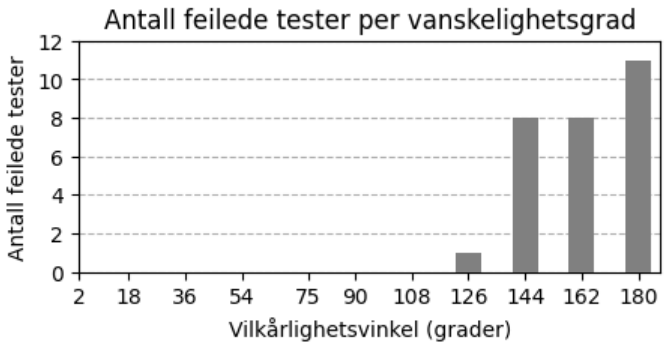
Figur 39: Resultat - Bane 0: Rett gjeting.

[5] https://bit.ly/gjeting-0

Bane 1 - Slak høyresving: Bane 1 tar for seg et scenario med èn slak høyresving. Fra skjermdumpen i figur 40 ser man at det gjetes fra oppe venstre, via punkt ca midt på banen, til nede høyre. Fra resultatene i grafen fra figur 41 ser man neste tilsvarende resultater som for bane 0, men med litt flere feilede tester ved de høye vilkårlighetsvinklene. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[6].



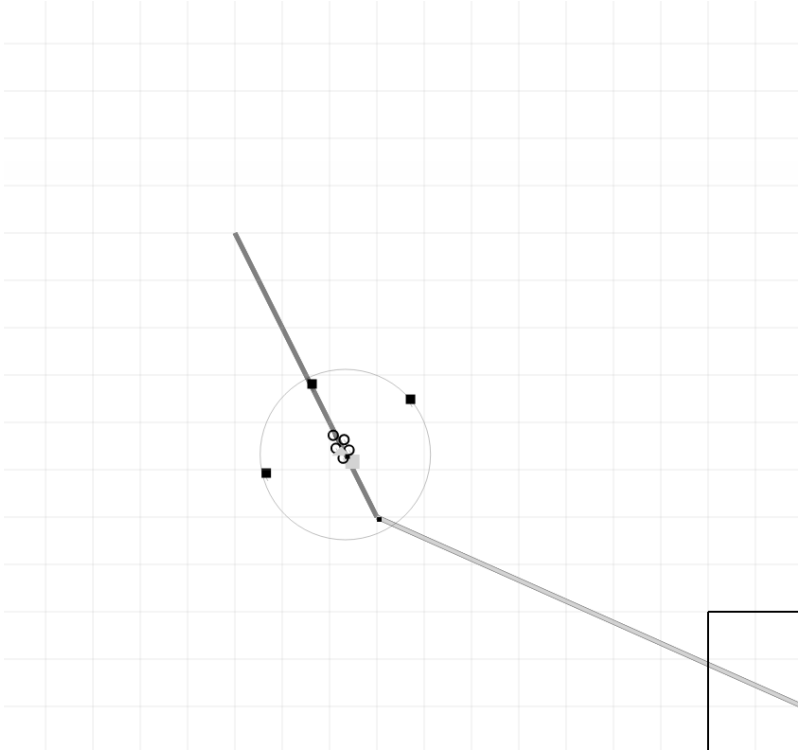
Figur 40: Resultat - Bane 1: Skjermdump slak høyresving.



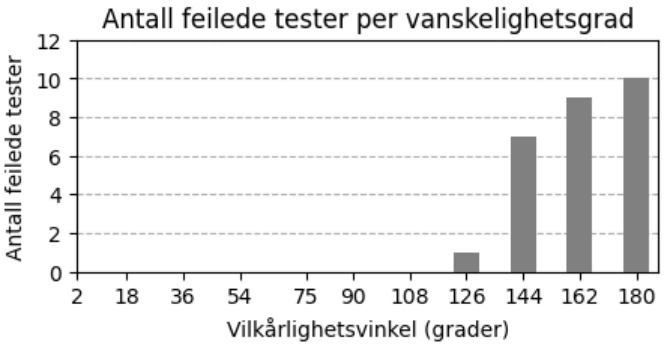
Figur 41: Resultat - Bane 1: Slak høyresving.

[6] https://bit.ly/gjeting-1

Bane 2 - Slak venstresving: Bane 2 tar for seg samme scenario som bane 1, men med motsatt sving. Fra skjermdumpen i figur 42 ser man at gjetingen svinges mot venstre i steden for høyre. Fra resultatene i grafen fra figur 43 ser man nesten identiske resultater som for bane 1. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[7].



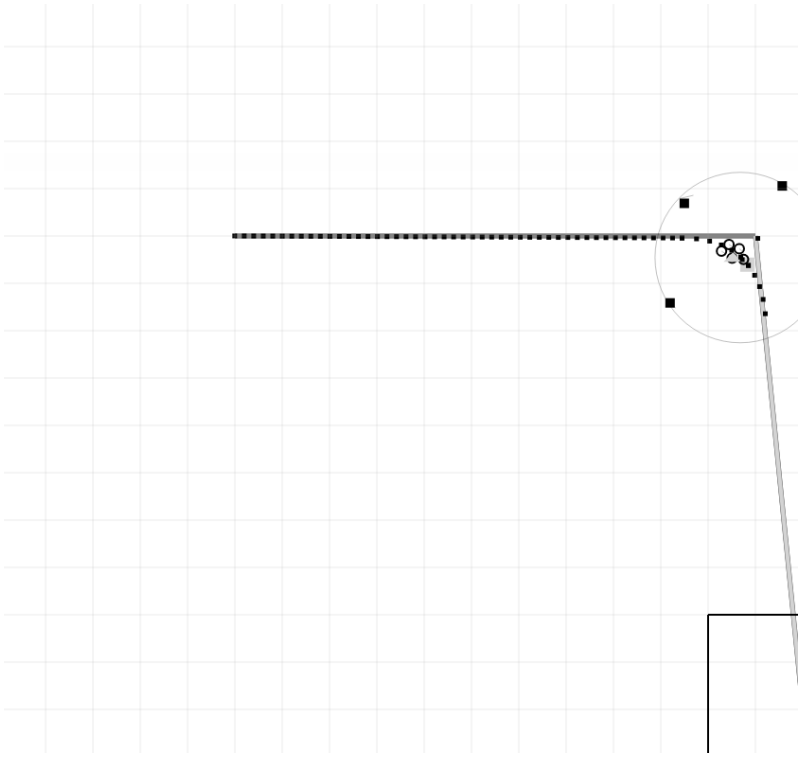
Figur 42: Resultat - Bane 2: Skjermdump slak venstresving.



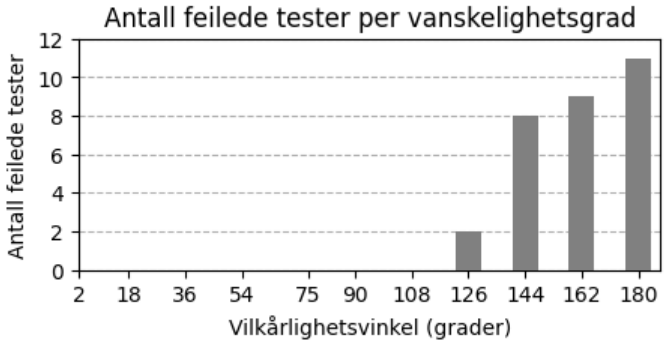
Figur 43: Resultat - Bane 2: Slak venstresving.

[7] https://bit.ly/gjeting-2

Bane 3 - Krapp høyresving: Bane 3 tar for seg et scenario med èn krapp høyresving. Fra skjermdumpen i figur 44 ser man at det gjetes fra oppe venstre, ut til høyre, deretter ned til målposisjon. Fra resultatene i grafen fra figur 45 ser man nermest identiske resultater som for bane 1 - slak høyresving. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[8].



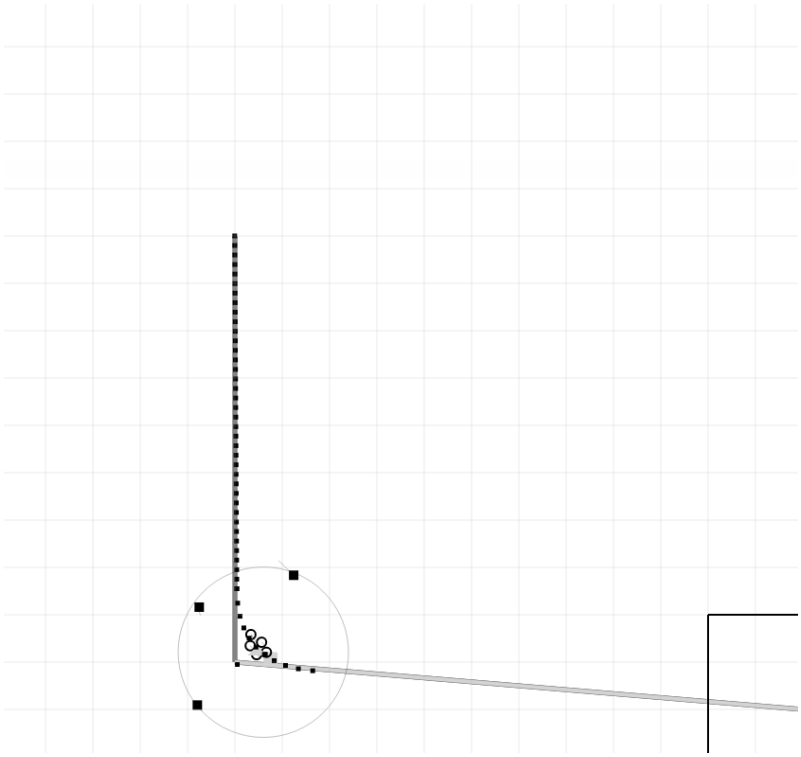
Figur 44: Resultat - Bane 3: Skjermdump krapp høyresving.



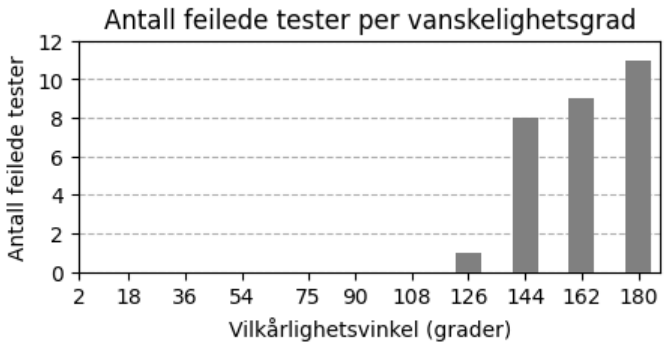
Figur 45: Resultat - Bane 3: Krapp høyresving.

[8] https://bit.ly/gjeting-3

Bane 4 - Krapp venstresving: Bane 4 tar for seg samme scenario som bane 3, men med motsatt sving. Fra skjermdumpen i 46 ser man at gjetingen svinges mot venstre i steden for høyre. Det gjetes dermed nedover først, derreter mot målposisjon i høyre hjørne. Fra resultatene i grafen fra figur 47 ser man tilnærmet identiske resultater som for bane 3. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[9].



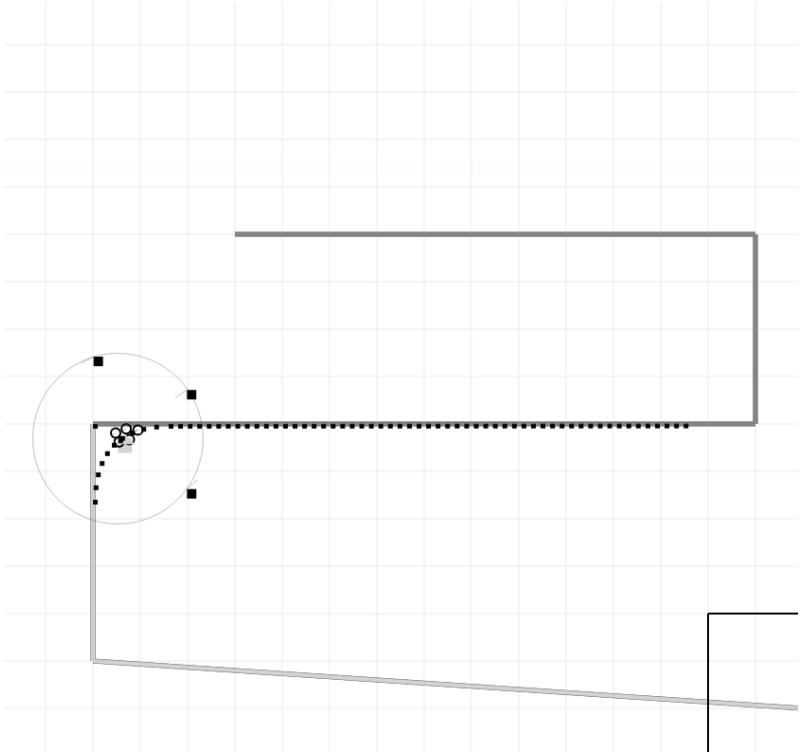
Figur 46: Resultat - Bane 4: Skjermdump krapp venstresving.



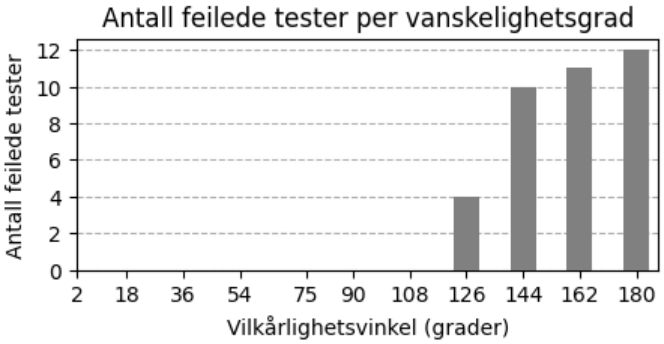
Figur 47: Resultat - Bane 4: Krapp venstresving.

[9] https://bit.ly/gjeting-4

Bane 5 - To u-svinger: Bane 5 tar for seg et scenario med to u-svinger, først mot høyre, deretter mot venstre. Fra skjermdumpen i figur 48 ser man at det gjetes først mot høyre i figuren, deretter mot venstre, før det gjetes mot målposisjonen nede til høyre. Fra resultatene i grafen fra figur 49 ser man noen flere feilede tester enn de forrige scenarioene, men vilkårlighetsvinkelen må justeres opp til 126° slik som for de andre scenarioene før det i hele tatt er feilende tester. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[10].



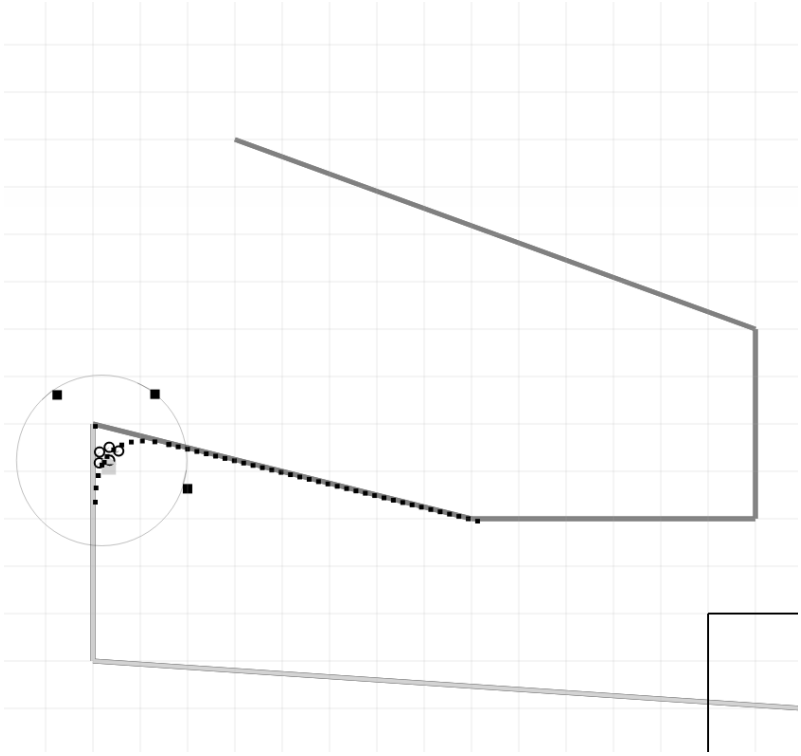
Figur 48: Resultat - Bane 5: Skjermdump to u-svinger.



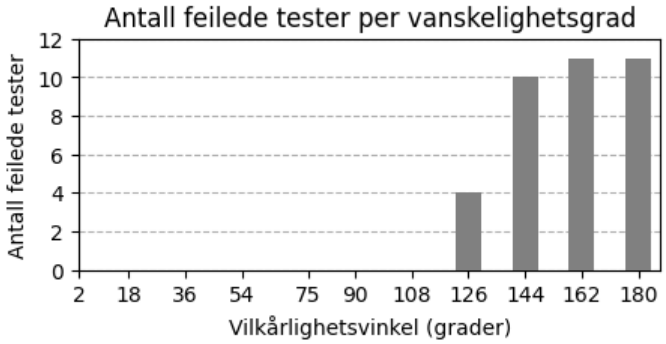
Figur 49: Resultat - Bane 5: To u-svinger.

[10] https://bit.ly/gjeting-5

Bane 6 - Krapp sving, under 90°: Bane 6 tar for seg tilnærmet det samme som scenariet i bane 5, men med en krappere sving. Fra skjermdumpen i figur 50 ser man at det gjetes tilnærmet på samme måte som i bane 5, men der hvor gjetingen foregår i skjermdumpen finnes det en krappere sving. Fra resultatene i grafen fra figur 51 ser man tilnærmet identiske resultater med bane 5, men med flere feilede tester for vilkårlighetsvinken 144° og 162°. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[11].



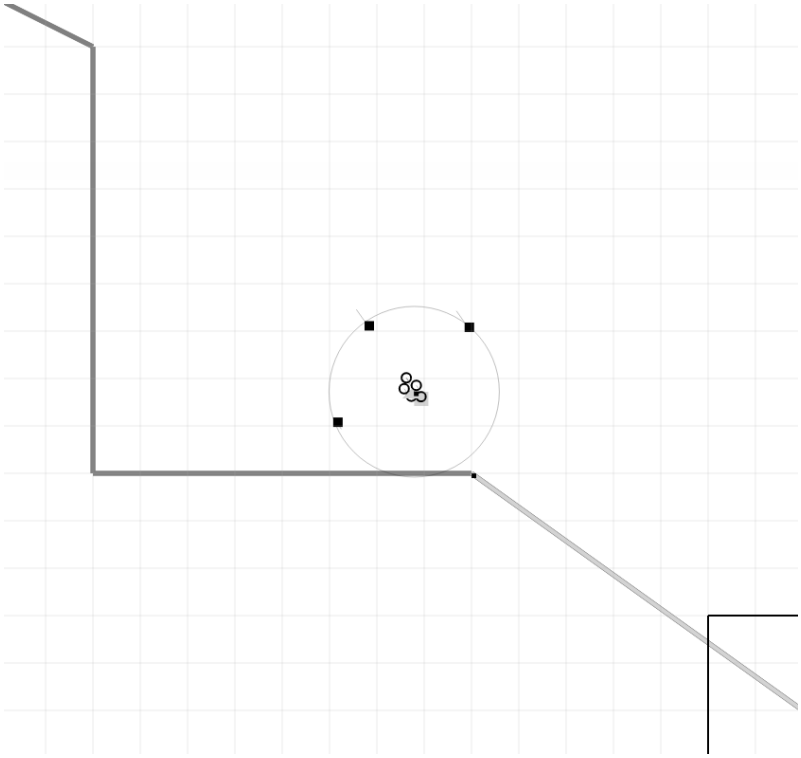
Figur 50: Resultat - Bane 6: Skjermdump krapp sving, under 90°.



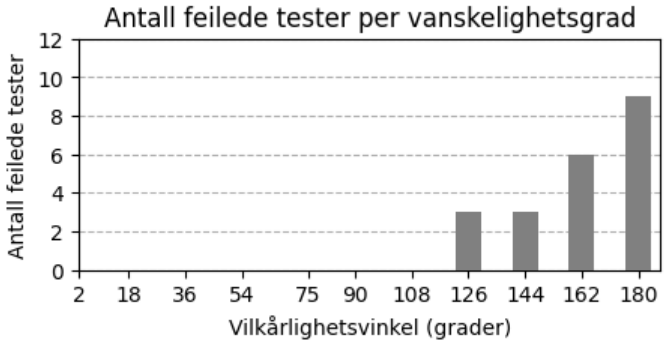
Figur 51: Resultat - Bane 6: Krapp sving, under 90°.

[11] https://bit.ly/gjeting-6

Bane 7 - Henting av flokk, rett linje: Bane 7 tar for seg et scenario med henting av flokk, via en rett linje til første punkt på banen. Fra skjermdumpen i figur 52 ser man at det gjetes fra oppe venstre, til nede høyre, via et punkt midt mellom. Sauene er altså på linje med banen som er korteste veien til målposisjonen. Fra resultatene i grafen fra figur 53 ser man få feilede tester. Helt opp til en vilkårlighetsvinkel på 162° er der lik eller under 50% feilende tester. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[12].



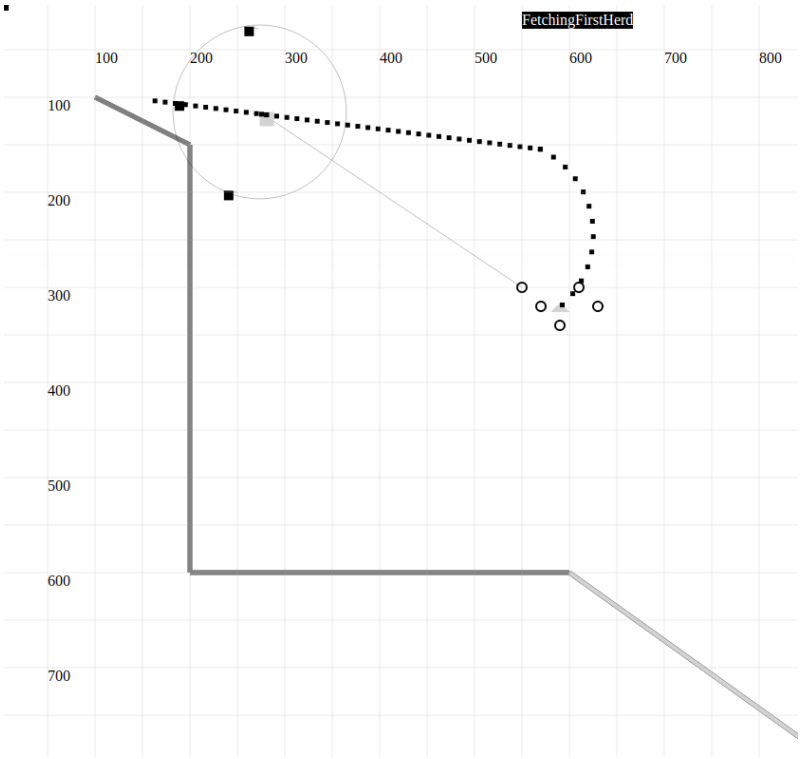
Figur 52: Resultat - Bane 7: Skjermdump henting av flokk, rett linje.



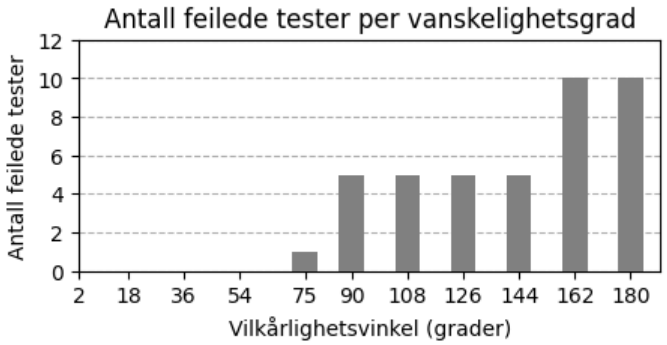
Figur 53: Resultat - Bane 7: Henting av flokk, rett linje.

[12] https://bit.ly/gjeting-7

Bane 8 - Henting av flokk, 135° vinkel: Bane 8 tar for seg et scenario med henting av flokk, via en slak sving som krever oppbygging av punkter. Fra skjermdumpen i figur 54 ser man at det er bygget opp en bane (de svarte små kvadratene) for å komme på baksiden av sauene før de gjetes mot målposisjonen via punktet (600,600). Fra resultatene i grafen fra figur 55 ser man feilende tester helt ned på en vilkårlighetsvinkel med 75°. Deretter er antall feilende tester stabilt fra 90° til 144°, før nesten alle testene feiler for resterende vilkårlighetsvinkler. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[13].



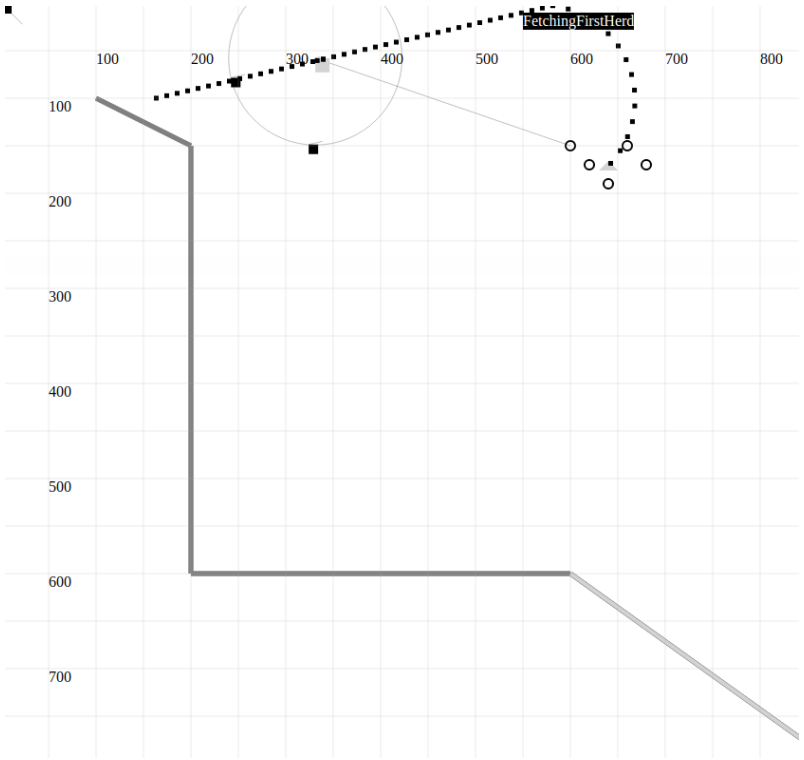
Figur 54: Resultat - Bane 8: Skjermdump henting av flokk, 135° vinkel.



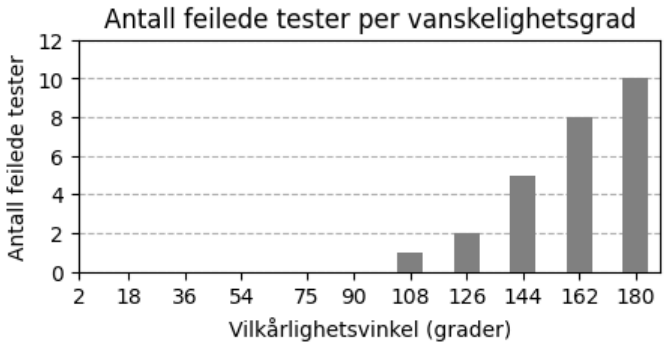
Figur 55: Resultat - Bane 8: Henting av flokk, 135° vinkel.

[13] https://bit.ly/gjeting-8

Bane 9 - Henting av flokk, 90° vinkel: Bane 9 tar for seg et scenario med henting av flokk tilsvarende som bane 8, men med en mindre vinkel mellom dronene, saueflokken og koordinatet de skal gjetes mot i det gjetingen starter opp. Fra skjermdumpen i figur 56 ser man at det er blitt bygget opp en bane (de svarte små kvadratene), for å komme på baksiden av saueflokken før de gjetes mot målposisjonen via koordinat (600,600). Fra resultatene i grafen fra figur 57 ser man at tester begynner å feile ved en vilkårlighetsvinkel på 108° og har stabil økning på resterende vilkårlighetsvinkler. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[14].



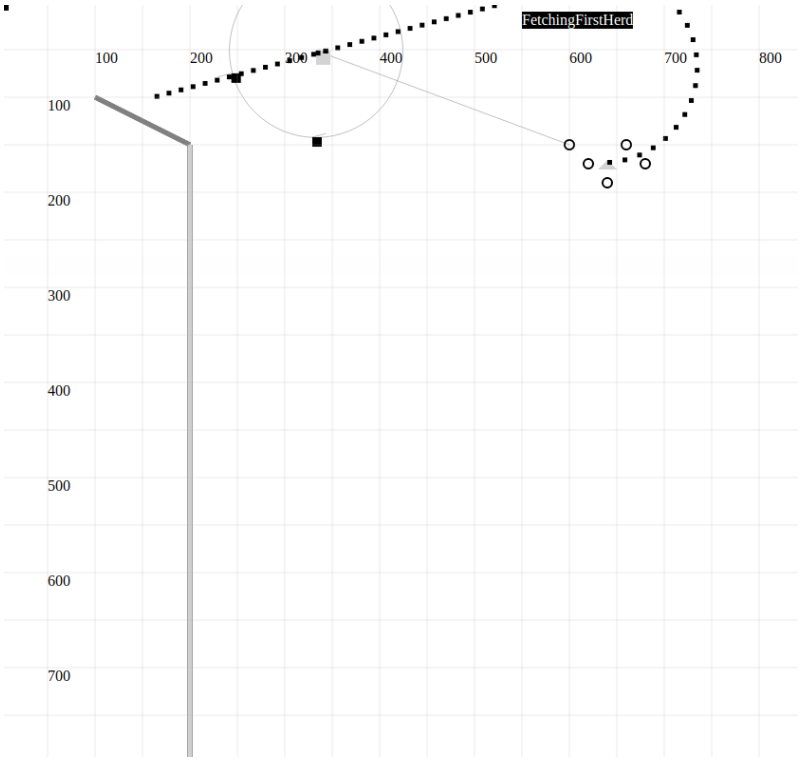
Figur 56: Resultat - Bane 9: Skjermdump henting av flokk, 90° vinkel.



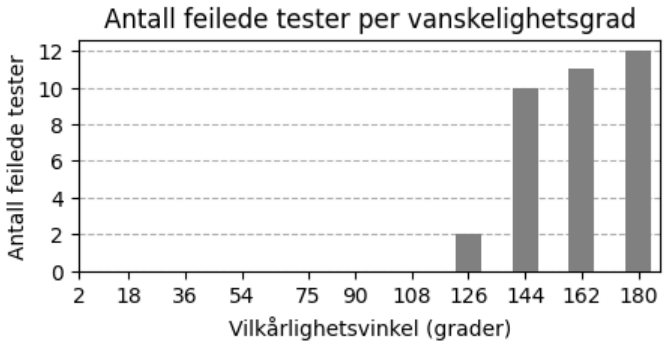
Figur 57: Resultat - Bane 9: Henting av flokk, 90° vinkel.

[14] https://bit.ly/gjeting-9

Bane 10 - Henting av flokk, 10° vinkel: Bane 10 tar for seg et scenario med henting av flokk tilsvarende som bane 8 og 9, men med en enda mindre vinkel mellom dronene, saueflokken og koordinatet de skal gjetes mot i det gjetingen starter. Fra skjermdumpen i figur 58 ser man at dronene starter i ca (150,100), henter saueflokken i (625,175) og gjeter dem mot (200,150). Fra resultatene i grafen fra figur 59 ser man at det først på vilkårlighetsvinkel 126° er noen få feilende tester, før det er drastisk flere på resterende vilkårlighetsvinkler. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 126° er publisert på denne linken[15].



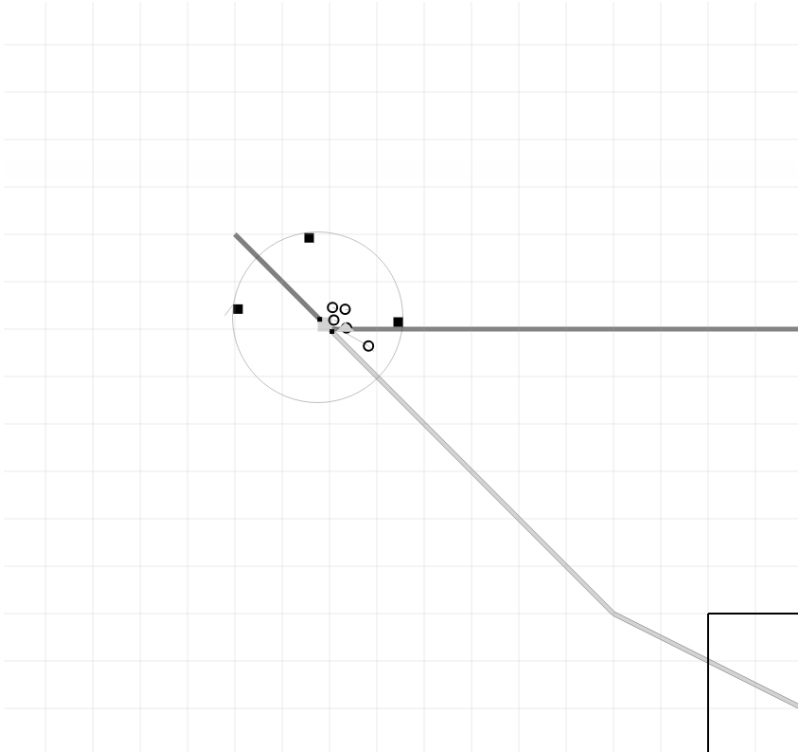
Figur 58: Resultat - Bane 10: Skjermdump henting av flokk, 10° vinkel.



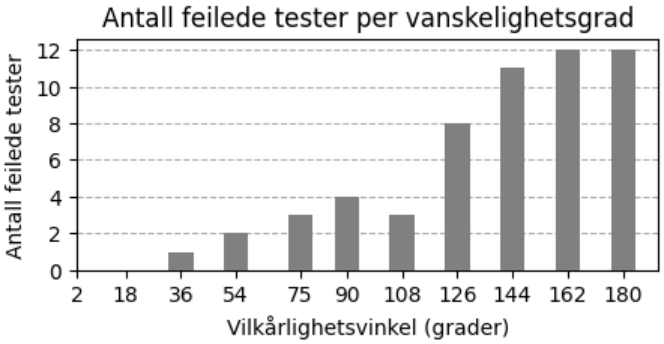
Figur 59: Resultat - Bane 10: Henting av flokk, 10° vinkel.

[15] https://bit.ly/gjeting-10

Bane 11 - Stikryss: Bane 11 tar for seg scenario med stikryss. Fra skjermdumpen i figur 60 ser man at det gjetes fra oppe venstre mot nede høyre. Det hvor gjetingen er i skjermdumpen finnes det er stikryss. Man ser at den ene gjeterdronen er sendt frem for å blokkere utgangen til venstre i krysset. Fra resultatene i grafen fra figur 61 ser man at det er feilende tester mye tidligere enn de andre banene. Allikevel er det først på vilkårlighetsvinkel 126° at over 50% av testene feiler. Video av scenariet med vilkårlighetsfrø 10 og vilkårlighetsvinkel 36° er publisert på denne linken[16].



Figur 60: Resultat - Bane 11: Skjermdump stikryss.



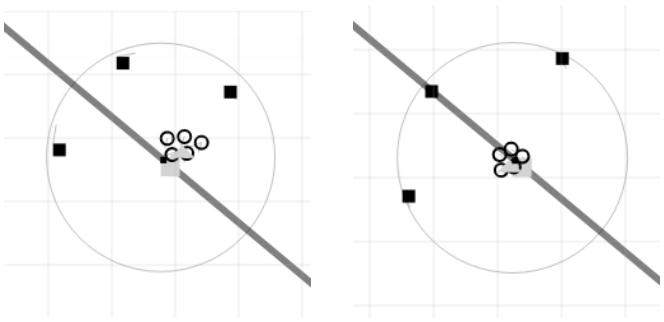
Figur 61: Resultat - Bane 11: Stikryss.

[16] https://bit.ly/gjeting-11

5.2 Posisjonering av gjetere

Fra kapitel 4.5.4 er det forklart flere funksjonaliteter som går på posisjoneringen av gjetingen. I dette kapitlet er disse funksjonalitetene listet opp med skjermdumper som visualiserer de forskjellige.

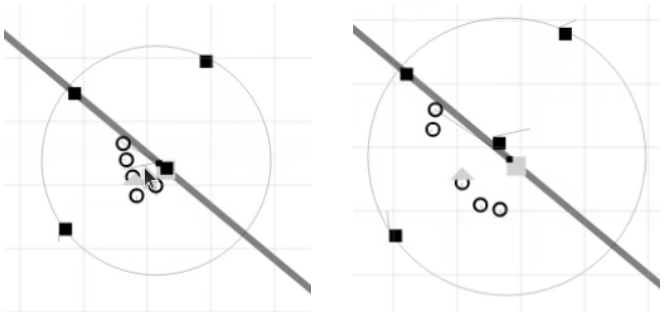
Justeing av vinkel for utspringer: Når en/flere sau er utenfor senter av flokken, justeres posisjoneringen av gjetere slik at den/de presses til senter av flokken. Figur 62 viser denne funksjonaliteten. Figur *a* til venstre viser justeringen av vinkelen, og figur *b* til høyre viser at justeringen fungerte etter hensikten; sauene samles på banen.



(a) Justert vinkel. (b) Utspringer hentet inn.

Figur 62: Resultat - Justering av gjeterposisjon for utspringer.

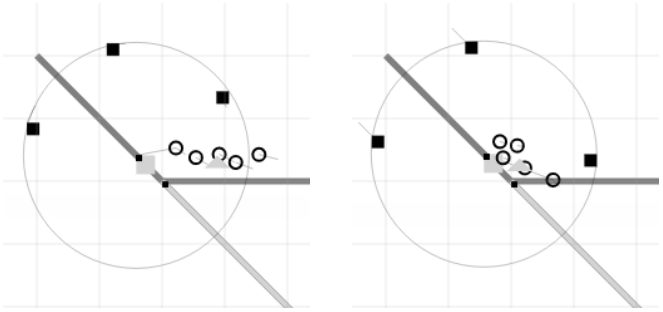
Økt flokkdiameter: Når flokkens diameter øker, øker diameter på gjeternes posisjon i forhold til oversiktsdrone. Figur 63 viser denne funksjonaliteten. Figur *a* til venstre viser normal diameter, og figur *b* til høyre viser utvidet diameter.



(a) Normal flokkdiameter. (b) Økt flokkdiameter.

Figur 63: Resultat - Økt flokkdiameter.

Stikryss: Når et stikryss nærmer seg, sendes nærmeste gjeter ut for å blokkere utgangen som ikke skal brukes. Figur 64 viser denne funksjonaliteten. Figur *a* til venstre er funksjonaliteten slått av. Sauene beveger seg i retning av utgangen som er feil. Her ser man også at funksjonaliteten for justering av gjetervinkel slår inn, men er ikke tilstrekkelig. Figur *b* til høyre er funksjonaliteten slått på. Sauene blokkeres og presses over i utgangen man ønsker.



(a) Stikryss uten blokkering. (b) Stikryss med blokkering.

Figur 64: Resultat - Stikryss.

6 Diskusjon

Utviklingen av kontrollalgoritmen med tilhørende virtualiseringsprogramvare ble utført i iterasjoner. I dette kapitlet er disse iterasjonene samt begrunnelse for valgene som er tatt forklart og diskutert.

6.1 Teknologi

Ved valg av teknologi ble behovet grundig vurdert. Oppgaven har ingen preferanse av teknologi. Det er utvikling av kontrollalgoritmen som er i fokus. Det ble vurdert teknologier med tanke på hva teknologien gir av fordeler til utvikling, for eksempel visualisering, ferdige matematiske metoder, og hvilken kompleksitet den drar med seg. Tar man for eksempel i bruk en 3D spill-simulator, kan den legge til rette for enkel visualisering, men det er som oftest en kunnskapsterskel å ta den i bruk, samtidig som det kan være utfordrende å justere noe i kildekoden, da den kan være flere tusen linjer. På den andre siden, kan det være veldig tidkrevende å bygge en visualisering. Fra oppgaven og dens avgrensning, er 2D visualisering ansett som tilstrekkelig for å kunne jobbe med kontrollalgoritmens funksjonalitet. Det er heller ikke noe krav til ytelse på visualiseringen, da den kun skal brukes i test- og demoformål. På bakgrunn av disse vurderingene er det valgt ren javascript og HTML for visualisering, samt en ASP.NET applikasjon som kjører kontrollalgoritmen og sender koordinater til visualiseringen.

6.2 Programflyt

Programflytene og hvordan koden eksekveres har utviklet seg i iterasjoner etter som behovene har kommet til. Fra starten av ble det satt opp enklest mulig programflyt. Denne flyten tillot kun en tilkoblet klient på webserveren. Andre som koblet seg til fikk opp samme gjetesesjon som første tilkoblet. Fra starten av var det initielle oppsettet tilstrekkelig, men etter hvert som det ble introdusert mer kompleksitet, som "hente flokk" og "stikryss" ble det hensiktsmessig å raskt kunne kjøre flere manuelle tester hyppig etter hverandre. Ved å videreutvikle programflyten til å støtte flere klienter, ble det mulig å koble til flere klienter med forskjellig testkonfigurasjon. Man fikk muligheten til å for eksempel sette opp mange tester på samme bane, men med forskjellig vilkårlighetsvinkel. Dermed ble testingen mer effektiv.

I utgangspunktet ble det tatt hensyn i alle kalkulasjonene hvor lenge det er siden forrige kalkulasjoner ble gjort, på grunn av at algoritmen kjøres på et operativsystem som ikke kan garantere real-time eksekvering. Dette betyr at operativsystemet kan kjøre kalkulasjonene ved ulikt tidsintervall, noe som gjør at kommandoene til dronene også kommer med forskjellige tidsintervall. Det er i utgangspunktet kritisk å sende kommandoene med like tidsintervall i forbindelse med at det er et styresystem for droner. Om for eksempel en drone skal akselererer kan kommandoene komme som

en sekvens på 10 m/s, 10,5 m/s, 11 m/s, 11,5 m/s, 12 m/s. For å få en stabil akselerasjon er det da kritisk at disse kommandoene kommer som en sekvens med like lang tid mellom hver kommando, eller at man tar hensyn til hvor lenge det er siden forrige kommando ble sendt, i kalkulasjonen. Det vil si at kommandoen justeres iht. til dette. Om utgangspunktet er at dronene skal akselerere med stigning på 1 m/s i sekundet, kan man sende kommandoene med et halvt sekund forsinkelse, slik som i eksempelet over. Om en kalkulasjon tar lengre tid å gjøre, f.eks om kalkulasjonen etter 10,5 er sendt tar 1 sekund, kan man i steden sende 10, 10,5, 11,5, 12, og man oppnår samme resultat.

Under utviklingen dukket det opp behov for å kunne reprodusere en test som feilet. Det vil si, at det måtte være mulig å kjøre en gjeting med samme konfigurasjon og få nøyaktig samme oppførsel. Behovet viste seg under manuell testing, men ble helt nødvendig når automatisk ende til ende testing ble implementert. Om en automatisk test feilet, ble det nødvendig å kunne kjøre nøyaktig samme test manuelt, for å finne ut hvor feilen lå. Dette var utfordrende å få til med eksekveringen og kalkulasjonene fra forrige avsnitt, og ble derfor skrevet om slik at ingenting har et forhold til tid, eller når noe ble kalkulert forrige gang. I steden for å ta hensyn til eksekveringstiden, ble det lagt på en forsinkelse før neste eksekvering starter. Denne forsinkelsen er kalt "virtualiseringsforsinkelse". Dette for å gjøre det mulig å kunne se hvordan gjetingen går for seg, da kjøring uten forsinkelse går så fort at det ikke er mulig. Denne forenklingen er hensiktsmessig for denne oppgaven da det hele kjøres i et testmiljø og at sannsynligheten er stor for at algoritmen skal kjøres i et real-time operativsystem når den settes inn i et fysisk system.

Den overordnede programflyten for eksekvering av kontrollalgoritmen er utelukkende satt opp for videre utvikling og test formål. Som nevnt i kapitel 4.2, er programflyten satt opp med fokus på å frakoble kontrollalgoritmen fra resten av programvaren, for eksempel simuleringen. Ved faktisk bruk av kontrollalgoritmen må den settes inn i en programflyt som kjører på oversiktsdronen.

6.3 Simulering av sauer

Ved valg av metode for å simulere saueflokken ble støtte for en iterativ utvikling hovedfokus. Dette vil si at man har sauer som kun blir påvirket av gjetere og ikke inneholder noen form for vilkårlige bevegelser i starten. Når kontrollalgoritmen ble i stand til å gjete disse sauene, kunne den bygges ut til å støtte sauer som for eksempel beveger seg litt i en vilkårlig retning under gjeting.

Tidlig i prosessen ble boids [6] metode vurdert. Dette er en maskinlæringsmodell som skaper flokk-oppførsel ved bruk av separasjon, sammenfallende bevegelse og samhold. Metoden tilbyr funksjonaliteten det er behov for i det ferdige produktet, men er ansett som overkomplisert for denne oppgaven, samtidig som det er dårlig støtte for å introdusere biter av kompleksitet.

Med inspirasjon fra boids [6] og artiklene i kapitel 3 ble matematikken for å simulere sauene implementert fra bunn. Den bygger på prinsippene fra inspirasjonen og ble utviklet i takt med kompleksiteten kontrollalgoritmen håndterer på i de gitte iterasjonene. Matematikken bak simule-

ringen baserer seg på vektorer og er ansett som overkommelig til å bli implementert fra bunn.

Kompleksiteten i simuleringen har vært gjennom en rekke iterasjoner, følgende er hovedtrekkene:

- Frastøt fra gjetere.
- Frastøt for å unngå at sauer er over hverandre.
- Holde sammen som flokk.
- Gressing (Skrotet).
- Bryte opp i flere flokker.
- Driv mot sti (Skrotet).
- Tidvis vikårlig retning under gjeting.
- Driv ut av stikryss.

Hovedtrekkene som fremdeles er implementer ble nærmere forklart i kapitel 4.3.

Gressing ble skrotet da det er en tilstand som ikke eksisterer slik gjetingen er implementert. Det enesete scenariet den eksisterer er akkurat i oppstarten av gjetingen, men den hadde så liten påvirkning at den er tatt bort, for å ikke ha med unødvendig kompleksitet. Driv mot sti under gjeting er ansett å gjøre gjetingen enklere. Da kontrollalgoritmen er i stand til å takle kompleksiteten som finnes, er denne forenklingen av gjetingen tatt bort.

6.4 Kontrollalgoritme

For å kunne teste teknologien og simuleringen av saueflokken, ble første iterasjon av gjeting implementert ved at flokken gjetes med musepekeren. Denne måten å gjete på gir full fleksibilitet når det kommer til manuell testing. Metoden ble aktivt brukt i alle iterasjoner i forbindelse med utviklingen av simulering av sauene. Ved å bruke musepekeren kan man få testet at sauene blir frastøtt når musepekeren nærmer seg, man kan presse sauene mot hvertandre for å teste at de ikke går over hverandre og så videre. I det ferdige produktet er ikke denne metoden nødvendig, men den gir rom for en mer iterativ test- og utviklingsprosess.

6.4.1 Banekoordinator

I tidlige iterasjoner av kontrollalgoritmen ble koordinatene lagret i en liste. Listen inneholdt alle koordinater i kronologisk rekkefølge fra banens startpunkt til sluttpunkt. Ved kontinuerlig behov for utvidet funksjonalitet, ble all relevant logikk lagt i en egen Banekoordinator modul. Dette ble gjort for å strukturere opp koden samtidig som å gjøre den mer testbar.

Ved introdusering av stikryss økte banekoordinatorens kompleksitet betraktelig. Datastrukturen gikk i hovedsak fra en en liste til et tre, eller potensielt en graf. Ved å se hver stikryss for seg, reduseres kompleksiteten fra en graf til et tre. Siden sauen blokkeres til å ta rett utgang, eller om sauen tar feil utgang feiler gjetingen, er kompleksiteten redusert til datastrukturen i figur 15.

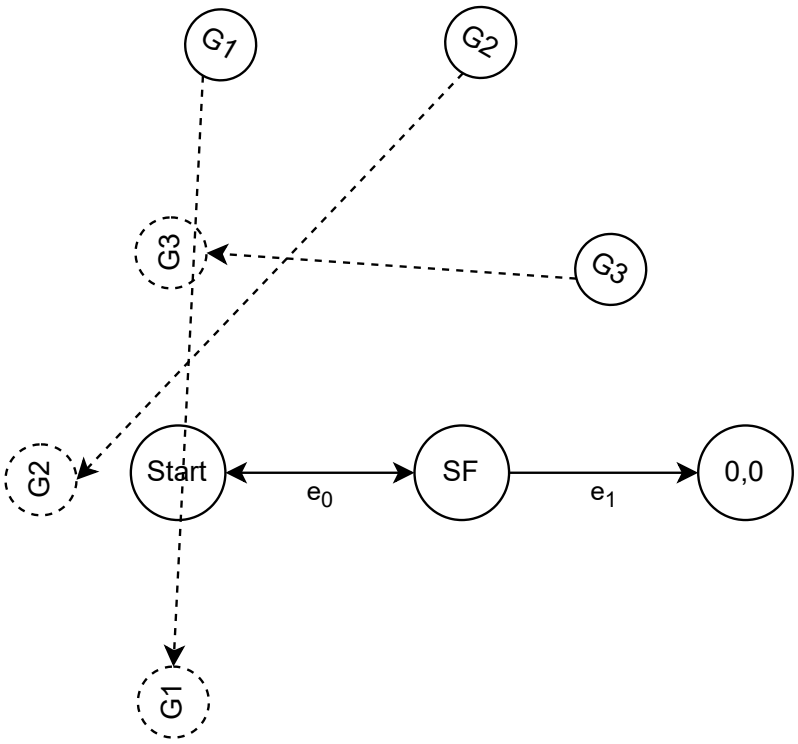
Første tilnærming til lagring av denne datastrukturen var en kø med par, hvor første element var koordinatet til dronene, mens andre paret var koordinatet til sauene. Om koordinatene var forskjellige indikerte det et stikryss. Denne metoden ble fort forkastet da det ble vanskelig å uthente informasjon fra køen.

Ved neste tilnærming ble det implementert en egen datastruktur, slik det tidligere er nevnt. Det kommer kompleksitet ved å implementere denne datastrukturen selv, men det åpner også opp full fleksibilitet. Kompleksiteten innebærer egne metoder for initiering og iterering gjennom datastrukturen. Fleksibiliteten åpner for eksempel opp til å kunne utvikle egen metode for å hente ut informasjon om neste stikryss nærmer seg, basert på avstand.

6.4.2 Punktkalkulator

Henting av flokk: Ved de første iterasjonene i utviklingen av kontrollalgoritmen ble henting av flokk ignorert i den form at gjeterne starter på baksiden av saueflokken, og saueflokken står ved første koordinat på banen. Dette ble gjort for å dele problemstillingen opp i mindre deler, som tillatter å løse del for del og gjør det mer håndterbart. Henting av flokk ble først introdusert ved enklest mulig scenario. Scenariet består av at gjeterne, sauene og første punkt på banen ligger etter hverandre, slik at dronene kan fly i en rett linje for å hente inn sauene. Dette scenariet introduserer tilstanden "Hent flokk" som ble bygget inn i kontrollalgoritmen ved dette tidspunkt.

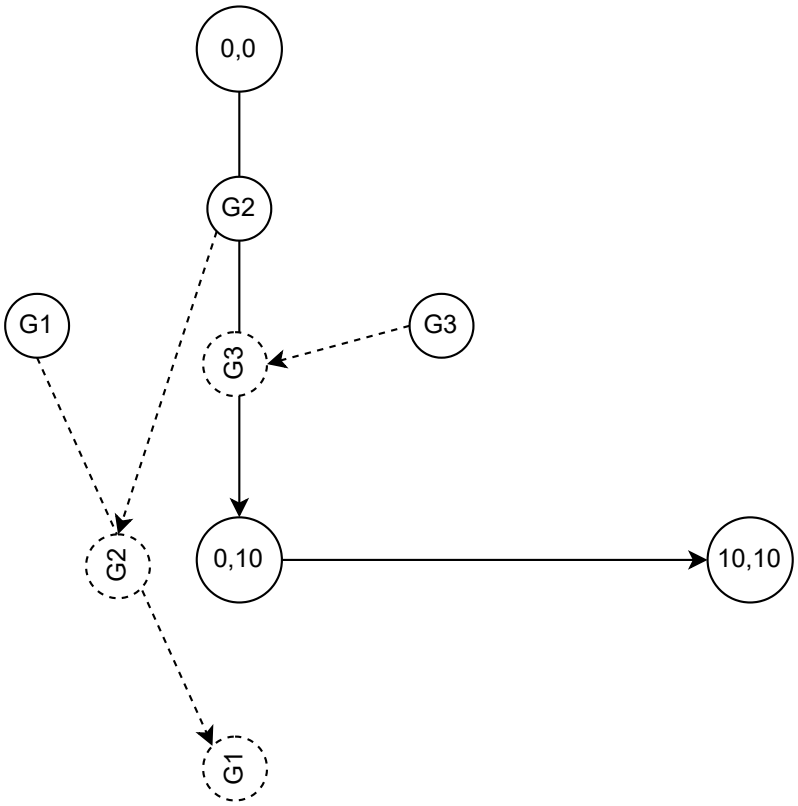
Ved neste iterasjon ble det forsket på å bygge opp en bane slik at dronene kommer på baksiden av saueflokken i tilfeller hvor sauene skal gjetes i retningen dronene kommer fra. Figur 65 visualiserer implementasjonen. Gjeterne er merkert med $G1-3$, saueflokken med SF og første punkt på banen sauene skal gjetes til med 0,0. Med dette utgangspunktet står gjeterne med under 90° vinkel på saueflokken i forhold til hvor de skal gjetes (0,0). Først kalkuleres e_1 ved å finne vektoren mellom saueflokkens senter og koordinat (0,0). Deretter kalkuleres e_0 ved å rotere retningen til e_1 med 180° og sette en fast lengde som gir dronene rom nok til å unngå sauene. Denne vektoren endte opp i et nytt start punkt for gjeterne. I stede for å fly rett til SF, fløy gjeterne til $Start$ i stedet. Som vist på figuren, fungerte implementasjonen til en viss grad, men når vinkelen mellom gjeterne, SF og (0,0) ble liten nok, det vil si at gjeterne kommer fra samme retning som (0,0), endte $G3$ opp med å påvirke SF før gjeteren nådde Start. Ved videre utvikling av denne metoden ble det lagt på en vinkel mellom e_1 og e_0 slik at Start kom nærmere gjeterne og man får en mer rundet start. Ved videre forskning ble det konkludert med at en mer generisk og enklere metode var å foretrekke. Det ble derfor introdusert en bane basert på en sirkel og en linje, slik det er forklart i kapitel 4.5.3.



Figur 65: Henting av flokk basert på vektor.

Det kan diskuteres om hele problemstillingen ved å bygge opp en bane for å komme på baksiden av saueflokken kan unngås ved at dronene flyr mye høyere inntil de når startpunktet. Denne tilnærming introduserer en annen type kompleksitet i form av høyder, vær og vind. Det er ikke gjort mer forskning rundt dette da den overnevnte løsningen er tilstrekkelig.

Gjeting: Ved de første iterasjonene av gjeting baserte styringen av dronene seg utelukkende på en vektor av gangen. Det vil si, en vektor skaper en kraft som drar dronene mot et punkt, helt til neste vektor trer i kraft. Figur 66 visualiserer utfordringen med denne type styring. Ved tiden $t = t_x$ er gjeterene $(G1 - 3)$ i posisjonen med heltrukkede linjer. Ved $t = t_{x+1}$ skal de være i posisjonen med stiplede linjer. Dette forårsaker gjeting av saueflokk rett frem, helt til de plutselig skal gjetes mot høyre i figuren. Resultatet er en unaturlig bane for saueflokken som resulterte i at sauer ble etterlatt.



Figur 66: Gjeting basert på vektor.

Etter flere iterasjoner med styring basert på vektorer, ble det bygget opp en mer naturlig bane ved bruk av flere punkter. Disse punktene ble kalkulert som punkter langs en bézierkurve, nærmere forklart i kapitel 4.5.3. Dette ble gjort i sammenheng med videreutvikling av tilstandsmaskinen, forklart i kapitel 4.5.5. Tilstandsmaskinen tillot å koble fra logikken som trengs ved kalkulering av bézierkurve fra gjetingen. Noe som forenklet å ta i bruk en bane med flere punkter.

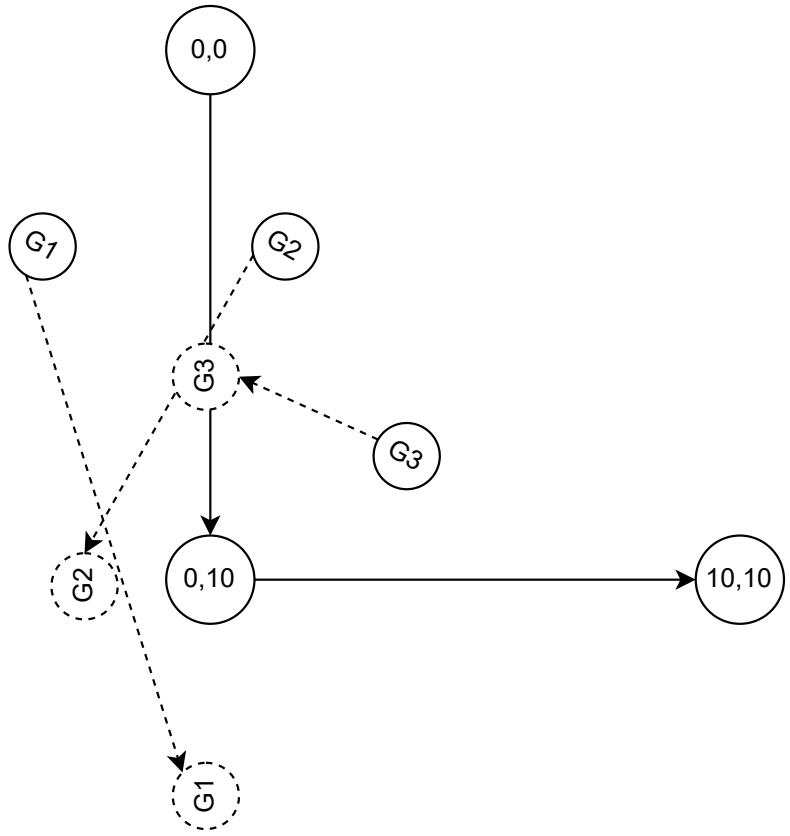
6.4.3 Posisjonering av gjetere

Metoden for å posisjonere gjeterne har vært gjennom flere iterasjoner. Etter hvert som banen og sauene ble mer krevende har posisjoneringen av gjeterne blitt videreutviklet. Hovedfunksjonaliteten som ble beholdt er forklart nærmere i kapitel 4.5.4 og kan oppsummeres som følgende:

- Statisk posisjonering på randen av sirkel.
- Automatisk justering av sirkel diameter.
- Automatisk justering av vinkel på flokk i henhold til utspringer.
- Blokkering av utgang i stikryss.

Statisk posisjonering på randen av sirkel var første tilnærming. Metoden er inspirert av metoden

brukt i [15]. I artikkelen justeres også kontinuerlig vinkelen dronene står på flokken, pluss/minus noen grader, uavhengig av flokkens oppførsel. Dette gir en god dekning for eventuelle utspringere. Denne kontinuerlige justeringen av vinkel er ikke tatt med i første omgang, da denne oppgavens gjeting har i hovedoppgave å gjete langs en bane med svinger. Ved å kontinuerlig justere vinkelen, viste det seg med testing at det har en negativ effekt, for eksempel ved en 90° vinkel, slik som på figur 67. Det gjetes fra topp til bunn i figuren og gjetingen er i ferd med å treffe en sving mot høyre i figuren, da sluttpunkt er (10,10). Ved kontinuerlig justering av vinkel, kan utgangspunktet bli som i figuren, det vil si, man jobber med å presse sauene i feil retning rett før svingen. Ved testing av denne funksjonaliteten resulterte det i at sauer ble etterlatt samtidig som det ble en mer kompleks gjeting i de tilfeller alle sauene ble med rundt svingen. Denne metoden med kontinuerlig justering av vinkel fra [15] ble derfor ikke videreført. Posisjonering av gjeterne på en sirkel rundt sauene viste seg ved testing å være et godt utgangspunkt for gjeting og ble implementert som fundamentet for posisjoneringen.



Figur 67: Potensielt utgangspunkt ved kontinuerlig justering av gjeter posisjons vinkel på saueflokken.

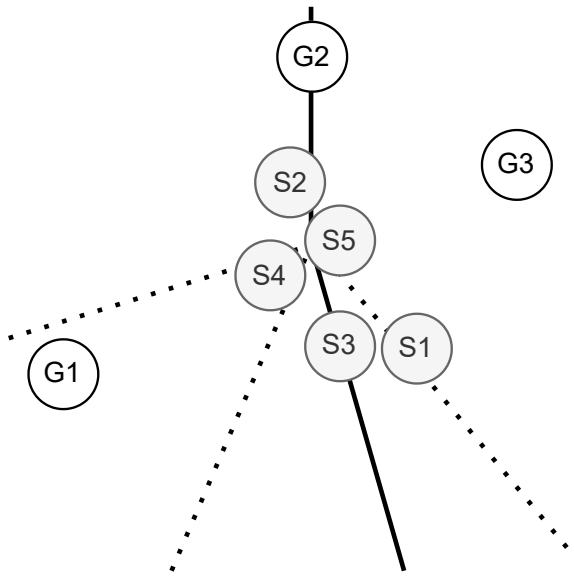
Som en første iterasjon i å forhindre utspringere, ble det implementert automatisk justering av gjeternes avstand til senter av flokken. Det vil si, diameteren til sirkelen gjeterne posisjonerer seg etter. Metoden er hentet og inspirert fra pausemekanismen i [8]. De har i utgangspunktet implementert pausemekanismen for å replikere hundens måte å gjete på. Ved testing av denne

metoden, viste det seg at når diameteren øker, stopper gjeterne litt opp, eventuelt flyr litt ut. Dette gir flokken litt pause. I denne pausen rekker kraften for flokkdannelse å tre i kraft, noe som resulterer i at utspringere sjeldnere slipper ut av flokken.

Etter hvert som kompleksiteten i sauene økte, startet testene å feile, sauer ble etterlatt. Det vil si at gjetingen ikke var i stand til å håndtere sauenes bevegelser. Det er i hovedsak "Tidvis vilkårlig retning under gjeting" det er snakk om på dette tidspunkt. For å imøtekomme denne utfordringen, ble det tatt inspirasjon fra funksjonaliteten forklart i forrige avsnitt. I steden for å justere vinkelen dronene har på saueflokken kontinuerlig, ble det implementert å justere den ved behov i henhold til utspringer. Dette ble nærmere forklart i kapitel 4.4 ved figur 27. Denne funksjonaliteten har samme utfordring som ble forklart i figur 67, men vil oppstå sjeldnere da den kun oppstår om det finnes èn eller flere utspringere samtidig som det er en vinkel på banen.

Ved introdusering av stikryss, økte kompleksiteten ved posisjonering av gjeterne betraktelig. I samarbeid med banekoordinatoren og tilstandsmaskinen ble dette løst. Banekoordinatoren ble utviklet til å tilby en metode som gav tilstandsmaskinen mulighet til å sjekke om ett stikryss nærmer seg. Når et stikryss nærmer seg, setter tilstandsmaskinen gjetingen i en undertilstand "Stikryss", forklart nærmere i kapitel 4.5.5 ved figur 31. Denne tilstanden indikerer til kontrollalgoritmen at nå skal gjeterposisjonene styres med egen logikk. Dette gjør at logikken for styringen i et stikryss kan være helt frakoblet logikken for gjeting. Dette gav full fleksibilitet og tillot å sende en gjeter frem for å blokkere utgangen i krysset sauene ikke skulle til. Logikken for blokkering av utgang er inspirert av domenekspert sin egen erfaring med gjeting og grundig testing.

I forbindelse med stikryss, er det ikke tatt noen betraktninger om krysset har flere utganger enn droner. Det trengs minst èn drone til å sørge for fremdrift, da er det to droner igjen til å blokkere utganger i stikrysset. Ved introdusering av flere utganger enn to, vil det være behov for ombygging av stikryss-manøveren. I steden for en direkte blokkering av utgangene vil det være hensiktsmessig å bruke to droner til å lede sauene mot rett utgang. Da ved å posisjonere dronene ut i fra vinkelen mellom inngang og den utgangen i strikrysset man ønsker at sauene skal ta, som vist i figur 68. I figuren er dronene $G1-3$ og sauene $S1-5$. Inngang og ønsket utgang er linjene i svart, fra topp i figuren til bunn. De stiplede linjene er utganger man ikke ønsker sauene skal ta. Man ser at $G1$ er posisjonert seg for å drive sauene bort fra utgangene til venstre i figuren, mens $G3$ til høyre i figuren. Dette i samarbeid med å senke farten på gjetingen vil i følge domene ekspert replikere menneskene sin metode i stikryss.



Figur 68: Stikryss-manøver - Flere utganger enn droner.

6.4.4 Tilstandsmaskin

Etter hvert som kontrollalgoritmen utviklet seg oppstod det tilstander i funksjonaliteten. For eksempel, var det behov for å kjøre en annen kode når flokken skal hentes, enn når den gjetes. Dette utviklet seg til en implisitt tilstandsmaskin, uten definerte rammer for hvilke tilstander som tillater hvilke overganger. For eksempel, ble det utfordrende å definere at når tilstanden er "Følg bane" skal det være umulig å gå tilbake til tilstanden "Hent flokk". På grunn av høyst iterativ utvikling og testing av flere tilstander ble det også uoversiktlig å holde kontroll på tilstandene. For å strukturere opp dette, ble det tatt i bruk en tilstandsmaskin i form av en programvarepakke. Pakken legger til rette for tydeligere rammer og struktur i programvaren, og dermed gjør den det enklere å holde kontroll i en programvare under kontinuerlig endring.

6.5 Gjeteområde og størrelser

Gjeteområdet, sauene eller dronene har ikke et forhold til fysiske størrelser. De er satt opp med relative størrelser, for eksempel kan en drone fly raskere enn en sau kan løpe. Slik er alle størrelser justert inn i forhold til hverandre. Ulempen med dette er skjulte avhengigheter, som gjør at man kan endre på en størrelse og da slutter noe annet å fungere optimalt lengre. Fordelen med dette er at man slipper å ha et forhold til fysiske størrelser, å gjøre kalkulasjoner slik at det skal stemme med fysikken. Dette er en forenkling som er gjort med tanke på hovedfokus på kontrollalgoritmen. Kontrollalgoritmen styrer utelukkende på koordinater og har ikke et forhold til størrelser. Utfordringer kan oppstå om for eksempel dronene ikke er i nærheten av å nå koordinatene før neste kommando kommer. Dette er ansett som en utfordring man må justere inn på det tidspunktet

fysiske droner blir tatt med i problemstillingen.

6.6 Testoppsett

En god måte å teste kontrollalgoritmen er en kontinuerlig utfordring. Programvaren kjører kontinuerlig og er derfor ikke like enkel å teste som en hendelsesbasert programvare. Det er for eksempel enklere å teste at en vare havnet i handlekurven når den skal, i kontrast med å teste at dronene flytter seg 10 piksler i x-retning og 13 piksler i y-retning når en av sauene beveger seg 5 piksler i x-retning relativt til saueflokkens senter, og når skal denne denne testen trigges? Dette har resultert i omfattende manuell testing. Det vil si at det gjøres en endring i koden, deretter kjøres en gjeting og man ser manuelt at endringen gav det ønskede resultatet. For å optimalisere den manuelle testingen har det i flere iterasjoner blitt gjort endringer i programflyten og måten kontrollalgoritmen eksekveres på. Dette for å kunne justere tiden det tar å kjøre èn gjeting, uten at det påvirker noen av kalkulasjonene som kan resultere i at dronene og sauene beveger seg anerledes. I tillegg er der laget funksjonalitet i brukergrensesnittet, som gjør det enklere å se at en gjeting kjører identisk med en annen, dette i form av å lagre banene til alle droner og sauer.

Etter hvert som store deler av kontrollalgoritmen fungerte for en gjeting, har det blitt laget ende til ende tester. Disse testene kjører en rekke scenarier for å teste ulik funksjonalitet i algoritmen og sjekker om sauene kom til målposisjon. Ulempen med disse testene er at man ikke får beskjed om hva som gikk galt, samtidig som man ikke kan teste enkelthendelser innenfor en gjeting. Man kan for eksempel ikke teste at dronene fulgte en sving slik det var tiltenkt.

Ende til ende testene er i hovedsak for å teste regresjon på gjete-nivå. For å oppnå regresjonstesting på hendelsesnivå kan man for eksempel logge pikslene til alle objektene for hver kjøring av kontrollalgoritmen, og sjekke at de er de samme neste gang man kjører testen. Dette kan også gjøres ved å sammeligne skjermdumper av hvor alle objektene har vært i løpet av en kjøring. Disse metodene hjelper dessverre ikke om man ønsker at en endring i koden har den rette effekten på gjetingen, fordi man ikke vet nøyaktiv hvilke piksler man kan forvente seg.

Funksjonaliteten som testes i ende til ende testene er noe overlappende. For eksempel testes funksjonaliteten i bane 0 også i bane 1, men utvidet til å teste slak sving. Dette grunner i programvarens iterative utvikling, hvor testene har vært en del av iterasjonene. Etter hvert som funksjonalitet har blitt implementert, har testene blitt utvidet med nye scenarier. Det er også hensiktsmessig og ha tester med overlappende funksjonalitet men ulik vanskelighetsgrad når noen tester feiler. Man kan for eksempel kjøre testene og så feiler bane 4 til og med 11, mens 0 til og med 3 kjører med suksess. Da får man en indikasjon på at det kan være noe galt i forbindelse med styringen i en krapp venstresving. Hadde programvaren blitt testet med en stor test som dekker alle scenarier, ville man i samme tilfelle fått en feilende test med mange flere potensielle feilkilder.

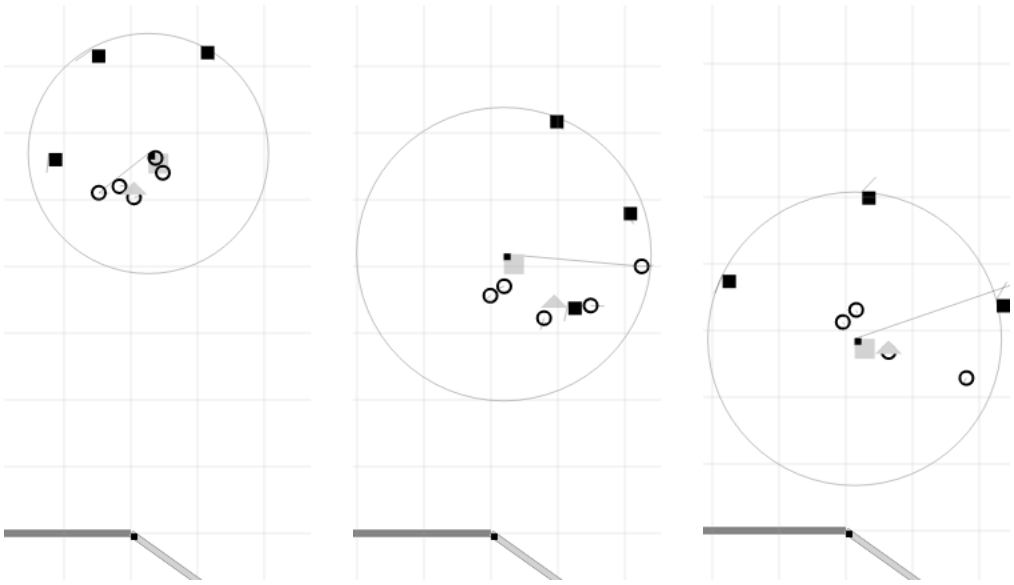
6.7 Problemstilling

I kapitel 2 er det definert en problemstilling med tilhørende forskningspørsmål. I dette kapitellet diskuteres det i hvilken grad og på hvilken måte disse er oppnådd.

Forskningsspørsmål 1: "I hvilken grad kan kompleksiteten ved å gjete en flokk sauer langs en sti håndteres av en kontrollalgoritme?", er i hovedsak testet på bane 0-6, selv om resterende baner også tester gjetingen i kombinasjon med andre scenarier. På bane 0 er gjeting langs en rett bane teste. Fra figur 39 ser man at gjetingen feiler på 1/12 tester først ved en vilkårlighetsvinkel på 126°. Uten ytre påvirkninger er det fornuftig å anta at saueflokken ikke plutselig vil endre retning med 126°. Videre ser man at det forkommer tilnærmet likt resultat på bane 1, 2, 3, 4, men antall feilede tester er noe høyere på 5, og 6, på samme vilkårlighetsvinkel. Bane 5 og 6 introduserer hyppige og krappe svinger, noe som forklarer en høyere grad av feilende tester. Ved hyppigere svinger er det mulighet for at saueflokken er ute av banen på grunn av utspringer eller forrige sving. Gjetingen kan da ha et dårlig utgangspunkt før neste sving og kan enklere feile. Videre feiler testene nesten identisk etter hvert som vilkårlighetsvinkelen justeres opp. I tillegg til manuell testing er dette en god bekreftelse på at simuleringen fungerer. Det vil si, det er mulig å justere opp vanskelighetsgraden på simuleringen så høyt at testene feiler.

Forskningsspørsmål 2: "I hvilken grad kan kompleksiteten ved å gjete en flokk sauer fra utmarken og inn på en sti håndteres av en kontrollalgoritme?", er testet på bane 7-10. På bane 7 testes det utregning av nærmeste punkt på sti uten oppbygging av punkter. Dette gir en rett gjetelinje og er derfor nesten identisk med bane 0. Bane 8 utmerker seg til å være vanskeligere også på lavere vilkårlighetsvinkler. Helt ned på 75° feiler en test. Deretter ligger det stabilt på 5 feilede tester helt opp til 144°. På høyere vinkel er resultatene tilnærmet tidligere tester.

For å finne ut en potensiell årsak til resultatene i bane 8, er det utført nærmere manuell testing med vilkårlighetsfrø 10. Det viser seg at innfallsvinkelen dronene har på saueflokken i forhold til koordinatet de skal gjetes til er et dårlig utgangspunkt. Dette resulterer i at det gjøres en korrigering for utspringer til venstre, deretter til høyre, som vist i figur 69. På figuren gjetes sauene mot det svarte punktet nederst i figurene. Fra figur *b* i figur 69 ser man at èn sau er i ferd med å unnslippe dronene. I figur *c* teller man kun 4 sauer. Denne oppførselen gjentar seg opp til og med vilkårlighetsvinkel 144°, deretter oppstår det flere vilkårlige feil. Det er også verdt å nevne at gjeterdiameteren her har en positiv effekt på gjetingen. Diameteren øker, noe som gjør at man får en pause i gjetingen og sauene blir ikke like raskt presset utenfor gjetingens bane. Dette er også visualisert i figur 69, hvor diameteren har økt fra figur *a* til *b*.

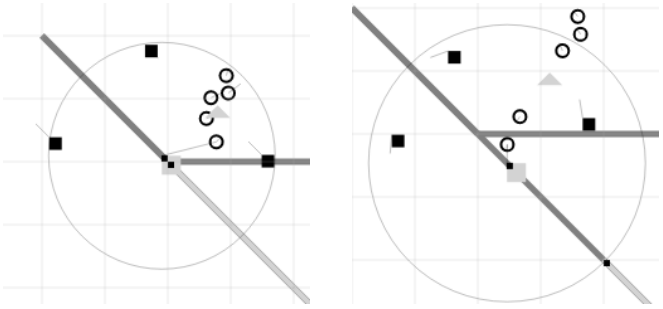


(a) Utspringer-manøver venstre. (b) Utspringer-manøver høyre. (c) Utspringer-manøver ferdig.

Figur 69: Uheldig utspringer-manøver.

Forskningspørsmål 3: "I hvilken grad kan kompleksiteten ved å få en flokk sauer til å velge rett utgang i et stikryss håndteres av en kontrollalgoritme?", er testet på bane 11. Som nevnt tidligere introduserer stikryss en ny dimensjon av kompleksitet. Det er en egen tilstand hvor vanlig gjeting går over til å sende den ene gjeteren frem for å blokkere en utgang. Testene for bane 11 begynner å feile allerede ved vilkårlighetsvinkel 36, og hele 4 av 12 tester feiler med vilkårlighetsvinkel 90. Deretter går antall feilende tester ned på 3 ved vilkårlighetsvinkel 108. Det antas at på grunn av faktorene av vilkårlighet, er dette en tilfeldighet. Det er ikke tatt med scenario hvor det finnes flere utganger enn droner til å blokkere de, som nevnt i kapitel 6.4.3.

For å finne ut en potensiell årsak til resultatene i bane 11, er det utført nærmere manuell testing med vilkårlighetsfrø 10. Figur 70 visualiserer hendelsesforløpet i det gjetingen feiler. Vilkårlighetsvinkelen samsvarer delvis med utgangen sauene ønsker, samtidig som gjeterne ikke justerer seg da de er i stikryss-tilstand. Når gjetingen bytter tilbake til gjete-tilstand er det for sent, da sauene allerede er for langt unna. Utvidelse av gjeterdiameteren hjelper heller ikke på det tidspunktet. Om de to siste gjeternes vinkel på utspringer hadde blitt justert, er det mulig at utspringerne kunne blitt stoppet og gjetingen fortsatt. Et annet alternativ til løsning kunne potensielt hvert å innført styring av farten. Da kunne farten ha blitt senket inn mot et stikryss for å ha bedre kontroll i krysset og latt sauene følge stien mere i steden for å bli gjetet.



(a) Stikryss med sauer på vei ut. (b) Etterlatte sauer.

Figur 70: Etterlatte sauer i stikryss.

Problemstilling: "I hvilken grad kan menneskelig sanking au sau erstattes med fire droner og en kontrollalgoritme?", dekker helheten til forskningspørsmålene. Den største utfordringen i oppgaven er å få alle delene til å spille sammen som en helhetlig sanking av sau. Samspillet er testet på alle banene i testoppsettet. For at en gjeting på en bane skal fungere må helheten fungere. Det er tilsammen kjørt 1584 ende til ende tester på gjetingen i en simulator som lar seg justere til gjetingen feiler. Det kunne forøvrig blitt kjørt enda flere tester med andre vilkårlighetsfrø, men med ett antall på 1584 tester, antas det at det gjennomsnittlige resultatet er oppnådd.

7 Konklusjon

Hovedmålet med oppgaven er å undersøke problemstillingen fra kapitel 2.2; "I hvilken grad kan menneskelig sanking av sau erstattes med fire droner kontrollert av en algoritme?". Det er utviklet en kontrollalgoritme for sanking av sau, i tillegg er det laget en simulering og et brukergrensesnitt for å visualisere hvordan kontrollalgoritmen utfører sankingen. Det er implementert en rekke metoder og moduler som er satt sammen til en komplett algoritme, nærmere forklart i kapitel 4, for å håndtere kompleksiteten i forbindelse med sanking av sau.

Problemstillingen er delt opp i tilhørende forskningspørsmål. Forskningspørsmål 1 lyder som følger: "I hvilken grad kan kompleksiteten ved å gjete en flokk sauer langs en sti håndteres av en kontrollalgoritme?" Kompleksiteten er den mest testede funksjonaliteten, hvor resultatene fra kapitel 5 viser til ingen feilede tester på de relevante banene, opp til og med vilkårlighetsvinkel på 108°. På bakgrunn av dette konkluderes det med at kompleksiteten ved å gjete en flokk sauer langs en sti, kan i høy grad håndteres av en kontrollalgoritme med tilhørende droner.

Forskningspørsmål 2 lyder som følger: "I hvilken grad kan kompleksiteten ved å gjete en flokk sauer fra utmarken og inn på en sti håndteres av en kontrollalgoritme?" Kompleksiteten er testet ved dekkende scenarier men har dårligere resultater ved henting av flokk i forbindelse med en vinkel på 135° mellom dronene, saueflokken og koordinatet de skal gjetes til. På bakgrunn av resultatene har det blitt utført utdypende manuell testing som forklarer resultatene. Innfallsvinkelen mellom dronene, sauene og koordinatet de skal gjetes til har et forbedringspotensialet, men det konkluderes med at kompleksiteten i forbindelse med gjeting av en flokk fra utmarken og inn på sti, kan i høy grad håndteres av kontrollalgoritmen. Innfallsvinkelen er ansett som en mindre utfordring som lar seg løse i løpet av få iterasjoner med utvikling.

Tredje og siste forskningspørsmål lyder som følger: "I hvilken grad kan kompleksiteten ved å få en flokk sauer til å velge rett utgang i et stikryss håndteres av en kontrollalgoritme?" Det er implementert en stikryss-manøver som håndterer blokkering av den utgangen sauene ikke skal til. Til forskjell fra de andre forskningspørsmålene, er dette i mindre grad testet, da det er satt opp et scenario på en bane, mens på de andre forskningspørsmålene er det flere. Det er også gjort forenklinger som gjør at scenariet kun dekker stikryss med to utganger. Det konkluderes med at kompleksiteten ved å få en flokk sauer til å velge rett utgang i et stikryss kan i høy grad håndteres av en kontrollalgoritme. Dette på bakgrunn av at forenklingen som er gjort ikke forenkler hovedutfordringen, den reduserer antall scenarier som kan oppstå i et stikryss. Testene viser at hovedmekanismen fungerer som den skal. Med ytterlig kapasitet til å gjøre flere iterasjoner med utvikling på mekanismen, samt inkludere flere mekanismer, som f.eks. fartskontroll, konkluderes det med at resterende scenarier kan håndteres.

Det er på bakgrunn av konklusjonene på forskningspørsmålene konkludert med at menneskelig sanking av sau kan i høy grad erstattes av fire droner kontrollert av en algoritme.

8 Videre arbeid

Kontrollalgoritmen i denne oppgaven støtter store deler av kompleksiteten ved sanking av sau, men det gjenstår fortsatt noe. I dette kapitlet er det diskutert videre arbeid og relevansen til det.

Som nevnt i kapitel 4.7 er det ikke tatt i betraktning reelle fysiske størrelsesforhold i denne oppgaven. Det er for eksempel ikke kalkulert inn en maks hastighet for sauene i meter per sekund i forhold til ekte sauer. Hastigheten er justert inn i forhold til de andre størrelsene. Det vil si at dronene er satt opp til å fly raskere enn sauene og så videre. Det er en hensiktsmessig forenkling, men for videre utvikling vil det være nyttig å ha en matematisk modell for simuleringen som står i samsvar med reelle fysiske størrelsesforhold. Da vil man lettere unngå at justering av en parameter påvirker andre parametre, siden man har et referansepunkt som er fundert i fysikken. Simuleringen vil i tillegg bli mer lik den fysiske verdenen, som gjør at man mest sannsynlig opplever mindre overraskelser når man tar i bruk algoritmen på et fysisk system.

For å få et mer virkelig gjetescenario vil det være nødvendig å introdusere kompleksiteten terreng innehar. Med terreng menes det i hovedsak høydeforskjeller. Det vil for eksempel være vanskeligere å gjete en sauflokk rett opp en bratt bakke, enn på flaten. Det kan også antas at sauene får en høyere fart, om de gjetes på flaten eller ned en bakke, i forhold til opp en bakke. Terreng drar også med seg kompleksiteten rundt områder det er umulig å gjete. Det kan for eksempel være vann eller en fjellside. Mye av kompleksiteten kan bli håndtert ved å legge opp en bane rundt slike områder, men med tanke på konsekvensene det kan medføre, vil det være fornuftig å ta hensyn til det i selve gjetingen. Ved å gjete en saueflokk rett mot en skrent kan sauene finne på å hoppe og i verste fall omkomme i fallet. For å introdusere kompleksiteten ved terreng, kan en potensiell fremgangsmetode være å introdusere et forhold til et kart på det gitte område det skal gjetes i. Banekoordinatoren kunne da hatt et forhold til kart og kalkulert eventuelle omveier i terrenget for å unngå vann eller fjellsider mellom to koordinater.

Som en videreføring av forrige avsnitt, kan det også være nyttig å introdusere vegetasjon. Det kan for eksempel være utfordrende å gjete sauer gjennom et område med tette einerbusker. Denne kompleksiteten kan på samme måte som terreng, muligens unngås ved at bonden setter opp en bane som unngår områdene med mye vegetasjon. En utfordring kan være at vegetasjonen kan forandre seg fra år til år og da kan det være krevende for bonden å holde seg oppdatert på hele området. For å ta hånd om kompleksiteten med vegetasjon, kan det for eksempel introduseres kart som inneholder denne informasjonen. For å ha et mer oppdatert bilde av nå-situasjonen kan man heller introdusere tolkning av områdets vegetasjon ved hjelp av oversiktsdronen. Det kan være mulig å gjøre nåtids omkalkulering av banen basert på denne tolkningen. Banekoordinatoren kunne tatt inn tolkningen eller vegetasjonskartet i forbindelse med kartleggingen av banen.

Ved gjeting gjennom norskt terreng, er det i følge domenekspert vanlig at det forekommer innhengninger. For å komme fra enn innhengning til en annen må sauflokken gjetes gjennom et le som

skiller de. Det er også vanlig at gjetingen avsluttes ved å gjete saueflokken inn i en innhengning slik at bonden kan laste sauene ombord i et transportmiddel. For å implementere denne kompleksiteten måtte sauene reagert med å bli frastøtt av innhegningens gjerde, utenom der hvor leet er. I tillegg er det rimelig å anta at dronene må gjøre en le-manøver slik som ved stikryss. Hva denne manøveren er, må det forskes på, men den underliggende mekanismen for å gjøre en manøver ut over den vanlige gjetingen er implementert i forbindelse med stikryss.

Under gjeting hender det at en og annen sau kan komme seg ut av flokken. Oppgaven har ikke logikk til å håndtere dette scenariet, ut over det å hente sauen i neste runde av gjeting. En strategi kan være å ikke la dette oppstå, men det kan ansees å være litt optimistisk siden sauer er dyr. Som videre utvikling av kontrollalgoritmen burde det tas hensyn til dette scenariet. En potensiell løsning kan være å stoppe opp gjetingen for å sende ut èn gjeter som henter inn sauen som forlot flokken. Det blir mer utfordrende om det er flere sauer som forlater flokken. Som nevnt tidligere, er det implementert logikk for å gjøre manøver utenom gjetingen. Dette er et tilfelle hvor denne funksjonaliteten kunne bli tatt i bruk. Da ved å implementere en egen logikk som kjører for en av dronene i det en eller flere sauer forlater flokken.

Ved sanking av sauer på høsten, finnes det som oftes flere en fem sauer, slik denne oppgaven tar utgangspunkt i. For videre arbeid vil det være nyttig å legge på en overordnet styringslogikk, som kan hente neste flokk etter første flokk er hentet. Tilstandsmaskinen legger til rette for denne type logikk, det kan være en potensiell løsning å legge til en betingelse som sjekker om det er flere flokker å hente og setter tilstanden tilbake til "hent flokk" etter første flokk er i målposisjon.

Med unntak av gjeternes økte gjeter-radius, forklart i kapitel 4.5.4, har gjeterne statisk fart. Dette betyr at det gjetes like rask på en rett strekning som i en krapp sving. Ved manuell testing har det vist seg at dronene kan fly i fra sauene. Ved implementasjon av en mer sofistikert fartskontroll, kan det være hensiktsmessig å senke farten i situasjoner det oppstår, eller kan oppstå utfordringer.

Som forklart kapitel 6.6, er det implementert regresjonstesting på gjete-nivå. Det vil si at man får beskjed om en gjeting ble fullført, eller feilet. For å støtte en videre iterativ utvikling av programvaren, men en bedre tilbakemeldingsløkke ved feil, vil det være hensiktsmessig å få vite hva som gikk galt i en gjeting. Dette kan f.eks gjøres ved at det lagres skjermbilder fra hver kjøring av testene. Om bildene er forskjellige har det blitt introdusert en endring i gjetingen. Ved å vise både før- og etterbilde vil utvikleren kunne se hva som endret seg. På denne måten får man mer kontroll på hva som endrer seg i utviklingens iterasjoner.

Sist og muligens åpenbart, bør kontrollalgoritmen settes inn i et helhetlig system med fysiske droner og sauer. Da kan man evaluere om antakelsene man gjør i simuleringen stemmer overens med virkeligheten. Det vil være hensiktsmessig å gjøre en iterasjon med testing på et fysisk system en gang innimellom de virtuelle for å evaluere og gjøre justeringer.

Referanser

[1] *.NET — Free. Cross-platform. Open Source.* https://dotnet.microsoft.com/en-us/. (Accessed on 02/11/2023).

[2] *05979: Jordbruksbedrifter med ymse husdyrslag, etter husdyrslag, statistikkvariabel og år. Statistikkbanken.* https://www.ssb.no/statbank/table/05979/tableViewLayout1/. (Accessed on 01/17/2023).

[3] *ASP.NET — Open-source web framework for .NET.* https://dotnet.microsoft.com/en-us/apps/aspnet. (Accessed on 02/11/2023).

[4] *Bézier curve - Wikipedia.* https://en.wikipedia.org/wiki/B%C3%A9zier_curve. (Accessed on 03/19/2023).

[5] *Bézier Curve. Understand the mathematics of Bézier... — by Omar Aflak — Towards Data Science.* https://towardsdatascience.com/b%C3%A9zier-curve-bfffdadea212. (Accessed on 03/19/2023).

[6] *Boids - Wikipedia.* https://en.wikipedia.org/wiki/Boids. (Accessed on 04/05/2023).

[7] *C Sharp (programming language) - Wikipedia.* https://en.wikipedia.org/wiki/C_Sharp_(programming_language). (Accessed on 02/11/2023).

[8] He Cai mfl. «Behavior-Based Herding Algorithm for Social Force Model Based Sheep Herd». I: *Electronics* 12.2 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12020285. URL: https://www.mdpi.com/2079-9292/12/2/285.

[9] *Canvas API - Web APIs — MDN.* https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. (Accessed on 04/23/2023).

[10] *dotnet-state-machine/stateless: A simple library for creating state machines in C# code.* https://github.com/dotnet-state-machine/stateless. (Accessed on 03/11/2023).

[11] *Herding 200 animals with 4 barking drones by the proposed method and the benchmark method - YouTube.* https://www.youtube.com/watch?v=KMWxrlkU6t0&list=PLXMQiMCZECaV8MVP0c2D1OoT9T44ObvDC. (Accessed on 02/20/2023).

[12] A. R. Hevner. «The Three Cycle View of Design Science Research». I: *Scandinavian Journal of Information Systems* 19.2 (2007).

[13] *Kestrel web server implementation in ASP.NET Core — Microsoft Learn.* https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-7.0. (Accessed on 02/25/2023).

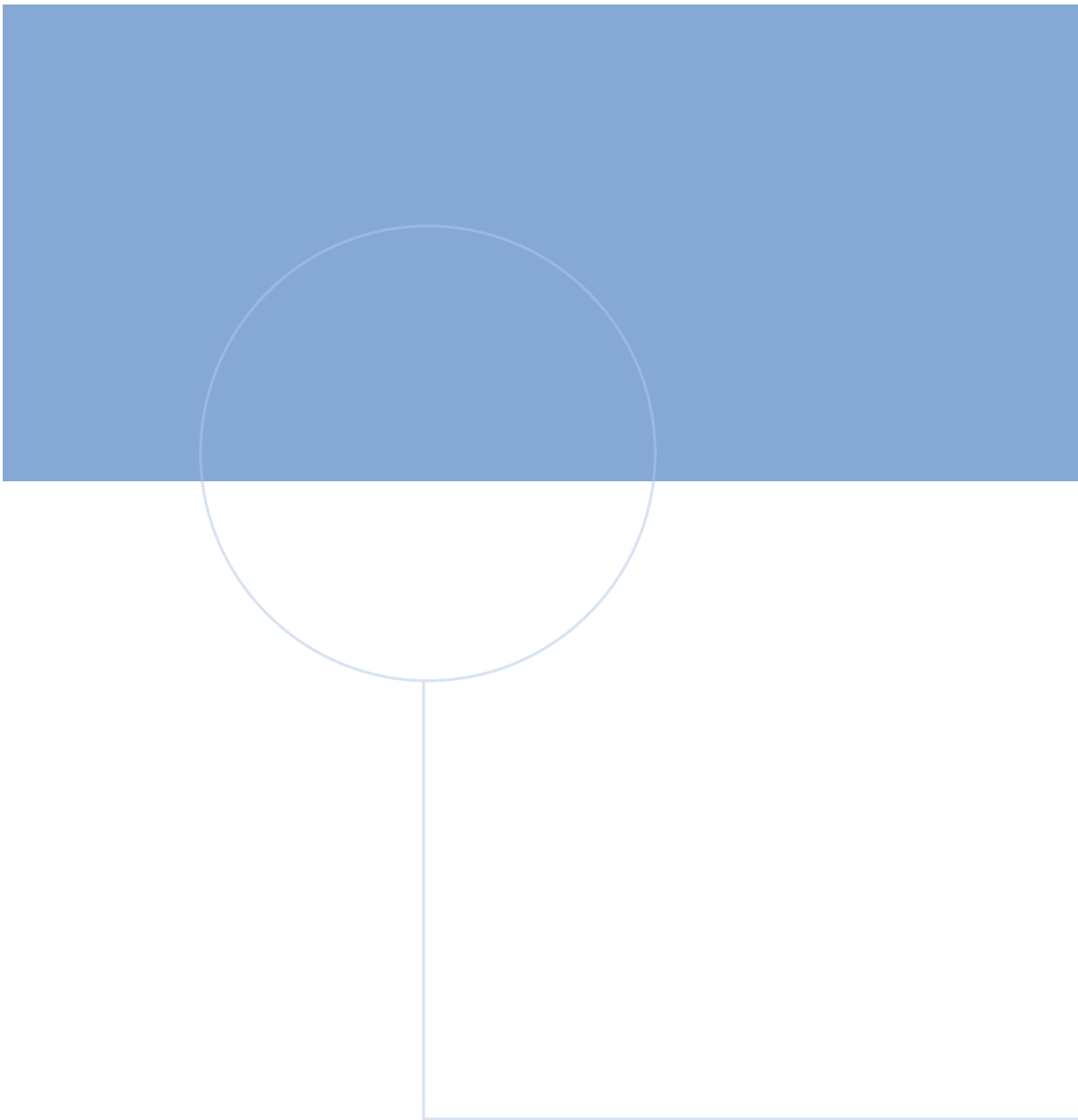
[14] Wonki Lee og DaeEun Kim. «Autonomous Shepherding Behaviors of Multiple Target Steering Robots». I: *Sensors* 17.12 (2017). ISSN: 1424-8220. DOI: 10.3390/s17122729. URL: https://www.mdpi.com/1424-8220/17/12/2729.

[15] Xiaohui Li mfl. «Robotic Herding of Farm Animals Using a Network of Barking Aerial Drones». I: *Drones* 6.2 (2022). ISSN: 2504-446X. DOI: 10.3390/drones6020029. URL: https://www.mdpi.com/2504-446X/6/2/29.

[16] Yu Liu mfl. «Sheepdog Driven Algorithm for Sheep Herd Transport». I: *2021 40th Chinese Control Conference (CCC)*. 2021, s. 5390–5395. DOI: 10.23919/CCC52363.2021.9549396.

[17] *Mattilsynets årsrapport for 2021*. https://www.mattilsynet.no/om_mattilsynet/mattilsynets_aarsrapport_for_2021.46704/binary/Mattilsynets%20%C3%A5rsrapport%20for%202021. (Accessed on 01/17/2023).

[18] *Real-time ASP.NET with SignalR — .NET*. https://dotnet.microsoft.com/en-us/apps/aspnet/signalr. (Accessed on 02/11/2023).



NTNU

Kunnskap for en bedre verden