



Användargränssnitt för systematiskt experimenterande
Coordination_oru

Maria Alkeswani
Datateknik Ingenjör

Student vid Örebro Universitet
Örebro 2023

Användargränssnitt för systematiskt experimenterande

Coordination_oru



Maria Alkeswani
Datateknik Ingenjör

Handledare: Franziska Klügl, Örebro University
Examinator: Pascal Rebreyend

Sammanfattning:

Coordination_oru är ett programramverk för forskning som skapades vid Örebro universitet i Sverige. Det är ett testramverk för en specifik algoritm för koordination av robotar som utvecklas vidare till en simulationsplattform som möjliggör systematiskt experiment. I det här examensarbetet skapas en experimentspecifikation med alla delar som behövs för att fullt konfigurera systematiska experiment för *Coordination_oru*-ramverket. Experimentspecifikation utvecklas för att anpassa ett grafiskt användargränssnitt som bidrar till att göra det enklare för användare att kontrollera, ändra och hantera systemet. Dessutom kan användare skapa och köra experiment med möjlighet att justera karta, väg, robotens hastighet, acceleration, storlek/form, färg och destination samt att se resultatet. Användargränssnittet har utvecklats med JSON-formatet för att hantera konfiguration av experimentspecifikation. Dessutom används CSV-formatet för att lagra resultatdata i tabellform under projektet.

Nyckelord:

Coordination_oru, User Interface, Användargränssnitt, Experimentspecifikation, Multi Robot systemet, Java, JSON, CSV.

Förord:

Ett djupt tacksamhet till min handledare Franziska Klügl för det exceptionella stödet som har givits under mitt arbete. Genom engagemang och expertis har bidragit till min utveckling som utvecklare och som person.

Till min fantastiska familj och älskade man, ett varmt tacksamma för det ovärderliga stödet ni har gett mig. Tack för er kärlek och uppmuntran som har varit avgörande för min framgång. Jag är evigt tacksam för er närvaro och stöd i mitt liv.

Innehållsförteckning:

1. Inledning :	5
1.1. Projekt background:	5
1.2. Projektidé	5
1.3. Projekt mål	6
2. Bakgrund	7
2.1. Coordination_oru-algorithm:	7
2.2. Simuleringsexperiment	8
2.3. Användargränssnitt:	9
3. Design	10
3.1. Översikt för strukturen:	10
3.2. Översikt för aktivitet:	11
3.3. Gränssnittsdesign:	12
3.3.1. En koncentrerad fönster	12
3.3.2. Explicit Komponent fönster	13
3.3.3. Flexibel gränssnitt	14
4. Metoder och verktyg	18
4.1. Metoder	18
4.2. Verktyg	19
4.2.1. Utvecklingsmiljö(IDL)	19
4.2.2. Coordination_oru-ramverk	20
4.2.3. Dataformat	20
5. Genomförning	22
5.1. Data för specifikation	22
5.2. Experiment klasser	23
5.3. JSON-fil:	26
5.4. Datainsamling under experiment:	28
6. Resultat	29
6.1. Öppna en JSON-fil:	29
6.1.1. Map sidan	32
6.1.2. Robots sidan	33
6.1.3. Experiment sidan	36
6.1.4. Användargränssnitt under experiment/test	37
6.2. Process om man börja med ny JSON fil:	42
7. Diskussion	44
7.1. Uppnående av projekt mål:	44
7.2. Gränssnitt-principer	45
7.3. Påverkan på samhället	45
7.4. Framtid arbete	46
8. Referenser	47

1. Inledning :

Denna rapport börjar med ett inledande kapitel som beskriver lite om projektets bakgrund, projekttiden och projektmålen.

1.1. Projekt background:

Coordination_oru är ett ramverk för rörelseplanering, koordinering och kontroll på multi-robotar som utvecklades av Federico Pecora och hans grupp vid Örebro universitet. Systemet erbjuder en lösning för att koordinera flera mobila robotar samtidigt. Den specifika funktionen av systemet är att det kan koordinera robotar som inte är kända från början och sen beräkna banor i realtid med hjälp av olika metoder som bestämmer förkörsrätt i konflikt situationer samtidigt med robotens rörelser. Systemet fungerar oavsett vilken rörelseplaneringsteknik eller kontrollsystem robotarna använder[7].

Systemet har också en enkel 2D-simulering av robotarna och en inbyggd rörelseplanerare som är baserad på OMPL-biblioteket. Verktöget levereras också med ett separat gränssnittspaket som möjliggör integrering med ROS och *navigation_oru* för en komplett lösning för multi-robot-koordinering och rörelseplanering.[7]

Coordination_oru garanterar att robotar kommer att nå sina destinationer. För att göra detta tar systemet hänsyn till flera faktorer till exempel robotar hastighet och acceleration.[7]

1.2. Projekttid

En experimentspecifikation kommer att skapas med alla delar som behövs för att kunna genomföra systematiska experiment med *Coordination_oru* ramverket. Ett grafiskt användargränssnitt ska göras för att ge användare tillgängligt tillgång att arbeta med experimentspecifikation, vilket innebär att det ska vara möjligt att konfigurera experiment visuellt.

För att utveckla ett användargränssnitt för koordinerade system behövs först och främst en klar förståelse för systemet, vilka delar som ska testas och vilka parametrar som ska justeras under experimenten. Det behövs också en tydlig bild av data som ska samlas in och hur de ska presenteras för användare.

1.3. **Projektmål**

Målet är att utveckla systemet med ett användargränssnitt för systematiskt experimenterande genom att:

- Identifiera och analysera de beståndsdelar som ingår i det koordinerade systemet som behöver specificeras, som kartor, vägar, uppdrag, robotar och andra komponenter.
- Skapa en experimentspecifikation med alla delar som behövs för att fullt konfigurera systematiska experiment.
- Utöka *Coordination_oru* för att automatisera experimentation baserad på experimentspecifikation.
- Implementera ett lämpligt användargränssnitt som gör det möjligt för användaren att skapa ett effektivt sätt att ange en enhetlig och komplett experimentspecifikation.
- Samla in vald data i en CSV-fil som beskriver rapporter om systemet, exempelvis hur ofta körning av system ska upprepas och när varje körning avslutas.

2. Bakgrund

Detta kapitel inleds med en introduktion av *Coordination_oru*-algoritmen för att ge en grundläggande förståelse av hur systemet fungerar. Målet med denna introduktion är att underlätta skapandet av experimentspecifikation genom att förstå *Coordination_oru*-algoritmen. Därefter kommer ett användargränssnitt för experiment att utvecklas, och det är viktigt att fokusera på kvalitet av användargränssnittet genom att utvärdera dess effektivitet, användarvänlighet och förmåga att möta användarnas behov och förväntningar.

2.1. *Coordination_oru*-algorithm:

Coordination_oru använder algoritmer för att hantera en kombination av rörelseplanering och koordination samt problem som kan uppstå vid rörelseplanering, koordinationer och kontroll på multi-robotar för att optimera prestanda och effektivitet i systemet. Vid rörelseplanering handlar det om att bestämma rätt rörelsebanor för varje robot för att undvika kollisioner med dynamiska och statiska hinder.^[1]

Koordinationer handlar om att säkerställa att robotarna samarbetar och kommunicerar effektivt med varandra för att utföra sina uppgifter utan att störa varandra. Kontroll handlar om att övervaka och reglera robotar-rörelserna för att de följer de önskade rörelsebanorna och utför sina uppgifter. Algoritmer som används för att koordinera rörelserna hos multi-robotar utformade för att hantera olika typer av robotar och miljöer, och de kan anpassas efter olika uppgifter.^[1]

En algoritm används för att koordinera rörelse av flera robotar för att undvika kollision. Den algoritmen tillämpas för att analysera områden där robotar ska passera för att nå sina destinationer. Varje steg samlar algoritmen in information om robotarnas position och beräknar banor som ska användas för att flytta dem från deras nuvarande position till målposition. Genom att analysera banor kan algoritmen identifiera de kritiska sektioner där robotarnas banor korsar varandra.^[1]

Den andra algoritmen bestämmer prioritet för robotar genom att ta emot banor, tid och tillstånd av nuvarande robotarnas läge och utvärdera deras tillstånd och beräkna distans till kritiska områden. Roboten som först ska ha tillgång till kritiska områden bestämmas efter sin prioritet.^[1]

Genom att förstå denna algoritm har fått insikt i hur systemet fungerar, vilket har hjälpt till att skriva specifikationer och utöka *Coordination_oru*.

2.2. Simuleringsexperiment

För att förstå och hantera svåra system är simulering ett viktigt verktyg. Simulering hjälper att uppskatta hur olika åtgärder påverkar systemet och analyserar hur det beter sig, vilket hjälper att få en bättre förståelse för komplexa system som en multi-robot system.[4]

En simulation är en process där man använder modeller och algoritmer för att efterlikna eller återskapa verkliga system eller händelser. Input representerar de data och parametrar som matas in i simulationen, medan output är resultatet som genereras av simulationen.

För att göra en simulation behöver man sätta in input som representerar de data och parametrar som matas in i simulationen, till exempel robots-hastighet, acceleration och storlek. Medan output är resultatet som genereras av simulationen, till exempel sluttid för experimentet, antal konflikter, robots-sluttid och väntetid .

Simulering kan användas för olika syfte, och det viktigaste syfte för det här examensarbetet är att stödja ingenjörsarbetet med system. Simulering kan användas för att validera och testa system och hjälpa till att förstå och förbättra systemets funktion. Det används även i planerings- och beslutsfattandeprocesser för att utvärdera olika alternativ och bedöma konsekvenserna av olika beslut.[4]

Simulationsexperiment ger viktig information om hur en modell beter sig. Det är viktigt att identifiera de viktiga komponenter och tillgänglig information under simuleringar för att kunna stödja automatisk återanvändning av simulationsexperiment. Dessutom måste simulationsexperimenten vara tydligt definierade för att kunna återanvändas. Genom att simulera olika situationer kan man observera hur modellen agerar och få en bättre förståelse för hur den fungerar och presterar.. Detta hjälper till att öka förståelsen om modellens funktion och vilka faktorer som påverkar dess funktion. När man utvecklar ett system görs simulationsexperiment ofta flera gånger för att se om vissa beteenden fortfarande stämmer efter ändringar. Metoder som automatiskt återanvänder och skapar simulationsexperiment kan hjälpa utvecklare att utföra simuleringar på ett mer ordnat och effektivt sätt.[3]

2.3. Användargränssnitt:

Examensarbetet fokuserar till stor del på att utveckla ett användargränssnitt för experiment. Det är därför nödvändigt att också beskriva bakgrunden om hur ska undersökas kvaliteten på ett användargränssnitt och ta hänsyn till de principer som bör tas i beaktning. I boken "Human-Computer Interaction" (tredje upplagan) av författarna Alan Dix, Janet Finlay, Gregory D. Abowd och Russell Beale, presenteras olika principer för att öka användbarheten och göra det enklare för användare att använda gränssnittet och få ut mesta möjliga nytta av det. Följande avsnitt kommer att introducera de viktiga principerna för detta arbete [8]:

- Flexibilitet : innebär att anpassa användargränssnittet till olika användares behov.
- Synlighet : innebär att göra tillämpliga funktioner och information tydligt synliga för användaren, vilket gör det lättare att upptäcka.
- Förutsägbarhet: handlar om att användaren kan utvärdera vad som kommer att hända baserat på deras tidigare erfarenheter av att använda systemet.
- Förtrogenhet: baserat på användarens kunskap och erfarenhet. Om användaren redan är bekant med liknande koncept eller funktioner kan snabbt förstå och använda det nya systemet.
- Konsistens: systemet visar sig liknande i liknande situationer, vilket skapar en förutsägbarhet i användarupplevelsen.
- Dialoginitiativ: innebär att ge användare utrymme att välja olika sätt att kommunicera med systemet.
- Flertrådig (multi-threading): handlar om att användaren kan arbeta med flera uppgifter samtidigt och växla mellan dem utan begränsningar.
- Observerbarhet: handlar om att användaren kan genom att titta på gränssnittet eller systemets förstå vad som händer i bakgrunden.
- Reaktionsförmåga: handlar om kommunikationen mellan användaren och systemet, en snabb och smidig kommunikation kan skapa en bättre användarupplevelse.
- Uppgiftöverensstämmelse: handlar om att systemet kan ge användaren de verktyg och funktioner som behövs för att slutföra sina uppgifter på bästa sätt.

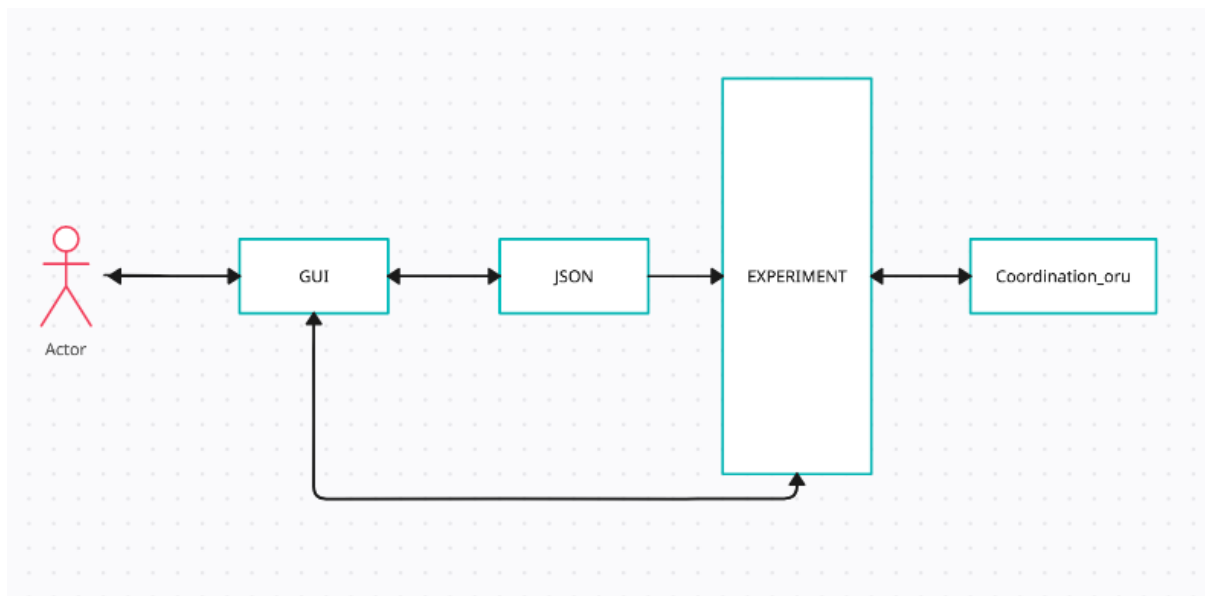
Många av dessa principer tas hänsyn till vid utveckling och diskuteras i kapitel 7.

3. Design

Detta kapitel innehåller tre olika delar/designer för att underlätta förståelsen av systemet och hur det fungerar. I den första delen beskrivs systemet och hur de olika komponenterna samverkar med varandra. Den andra delen förklarar processer och aktiviteter som inträffar under utvecklingen. Den tredje delen fokuserar på designen av gränssnittet.

3.1. Översikt för strukturen:

Detta komponentdiagrammet används för att beskriva strukturen av systemet och dess komponenter, samt deras relationer till varandra.



Figur 3.1 : Komponentendiagramm

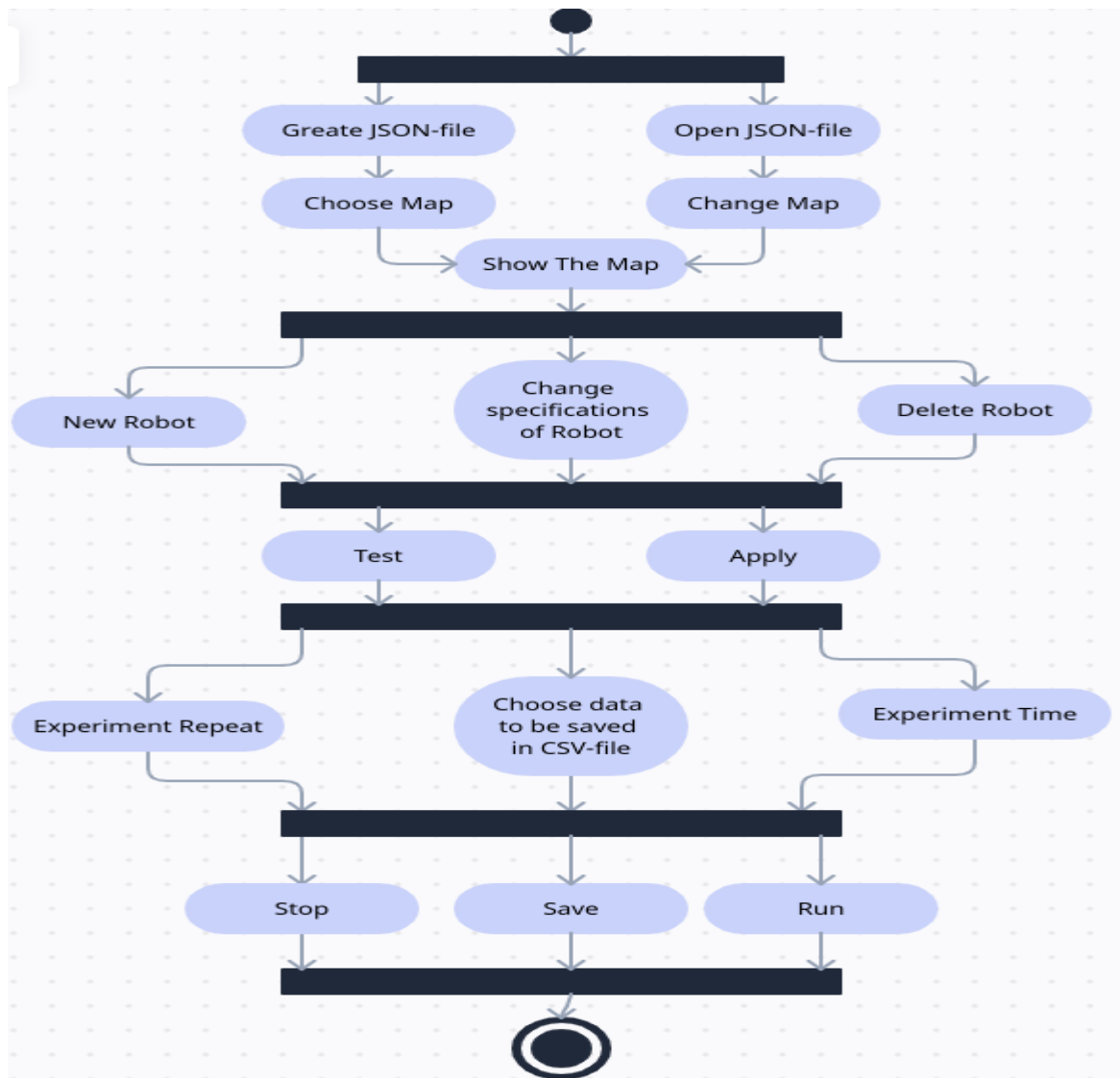
Komponenterna GUI, JSON, Experiment och *Coordination_oru* är sammankopplade på följande sätt:

- Actor och GUI: Actor är den personen som använder gränssnittet för att interagera med systemet, utföra uppgifter och/eller ändra data. GUI agerar som en output och visar användare data och/eller resultatet av systemet.
- GUI och JSON: JSON- fil är en strukturerad dataformat som används för att lagra data. GUI läser data från JSON- fil och visar den till användare, om användare ändrar data genom gränssnittet så uppdaterar JSON- fil med de nya informationen. Användare kan läsa från JSON- fil och skriva på JSON- fil genom gränssnittet.

- Experiment med GUI, JSON och *Coordination_oru*: experiment är huvudkomponenten i systemet som kan läsa från JSON-fil, analysera och specificera de beståndsdelar som ingår i det *Coordination_oru* samt interagera med användargränssnittet när data användas, ändras, sparas och den kan köra experiment i

3.2. Översikt för aktivitet:

Activity diagram används här för att beskriva hur systemet vägleda användare i vilken ordning aktiviteter beror på andra och bör utföras steg för steg i systemet. Det gör det smidigare att förstå aktiviteterna på ett tydligt sätt.



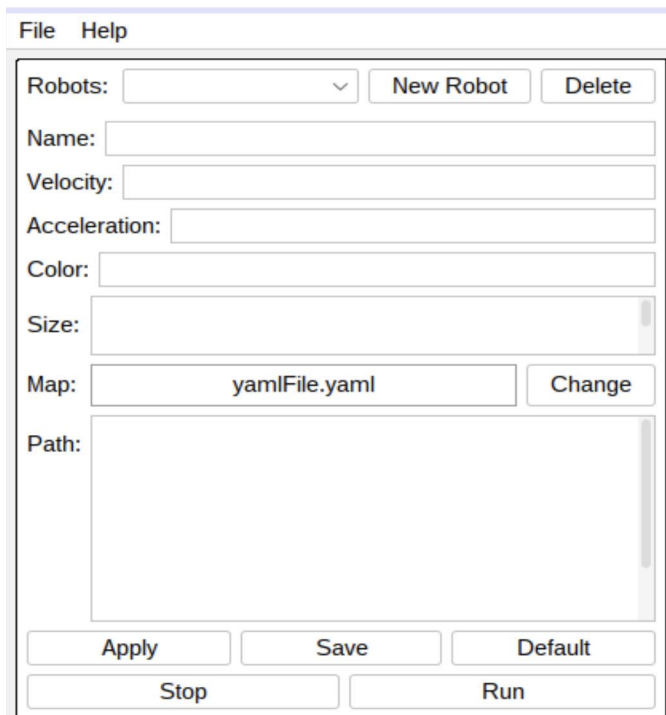
Figur 3.2 : Aktivitidiagram

Processen börjar med att användare skapa en JSON-fil, där det finns möjlighet att välja en karta eller öppna en befintlig JSON-fil med möjlighet att ändra på kartan. Därefter visas kartan för användare. Nästa steg är att hantera robot, användare kan skapa en ny robot, ta bort eller ändra specifikationerna för en befintlig robot. Efter att kartan och robotarna hanterats får användare tillämpa och testa systemet. Slutligen får användare hantera experimentetid och antal upprepningar samt lagringen av data i en CSV-fil. Därefter kan simulationsexperimentet köras och stoppas och resultatet sparas.

3.3. Gränssnittsdesign:

Det är viktigt för användare att skapa en bra gränssnittsdesign eftersom den påverkar hur enkelt och effektivt de kan interagera med systemet. Om gränssnittet är lätt att använda, kommer användarna att känna sig mer bekväma och det hjälper dem att uppnå sina mål snabbare och med färre misstag. Ett bra gränssnitt leder till att öka användarens förtroende för systemet. Därför analyseras tre typer av gränssnitt. Därefter kommer det bästa gränssnittet att väljas och implementeras.

3.3.1. En koncentrerad fönster



Figur 3.3 : Koncentrerad fönster

Det första alternativet är ett enkelt gränssnitt som har en okomplicerad design och fokuserar på att visa de grundläggande funktionerna för användare i samma fönster, vilket gör det enkelt att komma till varje komponent. I detta gränssnittet finns bara komponenter för Robot, Mapp med möjligheter att skapa en ny robot, ändra på robots data, välja en annan karta, testa och spara data.

Detta gränssnitt kan vara lämpligt för små och enkla system där användaren behöver utföra enkla uppgifter eller göra små ändringar i data. Det är dock inte lämpligt för detta arbete då det är svårt att utöka gränssnittet med helt nya komponenter. Gränssnittet har utvecklats och uppdelats i flera delar baserat på dess funktioner i det nästa alternativet.

3.3.2. Explicit Komponent fönster

The interface is titled "Explicit Komponent fönster" and features a menu bar with "File" and "Help". It is organized into three main panels:

- Robots Panel:** Contains a "Robots:" dropdown menu, "New Robot" and "Delete" buttons, and input fields for "Name:", "Velocity:", "Acceleration:", "Color:", "Size:", and "Path:". An "Apply" button is located at the bottom of this panel.
- Map Panel:** Located at the top right, it includes a text input field with the value "yamlFile.yaml" and an "Open" button.
- Experiment Panel:** Located at the bottom left, it contains buttons for "Save", "Save As", "Default", "Stop", and "Run".

A large "Log" area is positioned on the right side of the interface, currently displaying no content.

Figur 3.4 : Explicit komponent fönster

Andra alternativ för gränssnitt är mer utvecklat än första alternativet och har explicit ytan för varje komponent med flera komponenter som Log och Experiment. Men det är fortfarande ett fönster som är mer ordnat eftersom det är uppdelat i fyra avsnitt och varje avsnitt har sina egna delar.

Detta gränssnitt visar all information samtidigt på en plats och har flera funktioner och alternativ för användare att välja mellan på samma plats. Men detta kan orsaka vissa problem som till exempel att användare ändrar data medan systemet körs och det blir för stora fönster.

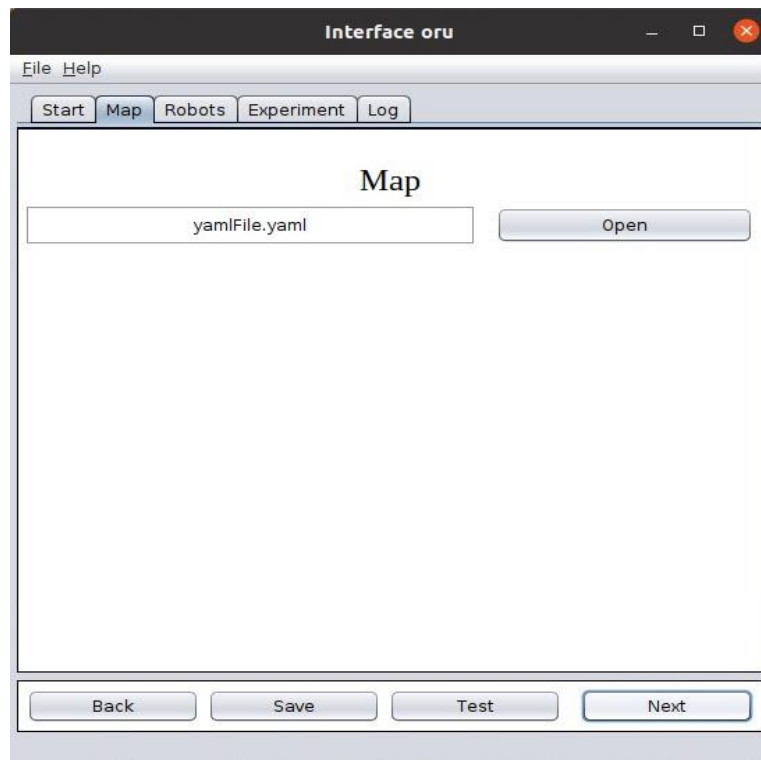
Detta gränssnitt har utvecklats till ett flöde av fönster i tredje alternativ.

3.3.3. Flexibel gränssnitt

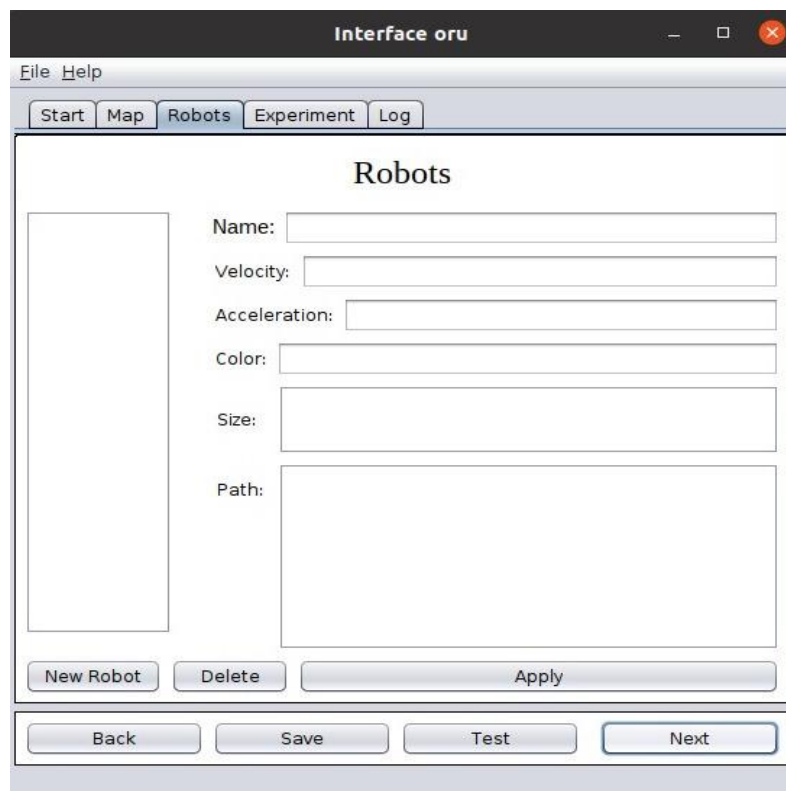
Den tredje option för gränssnitt består av flera fönster som vägleda användaren genom hela processen.



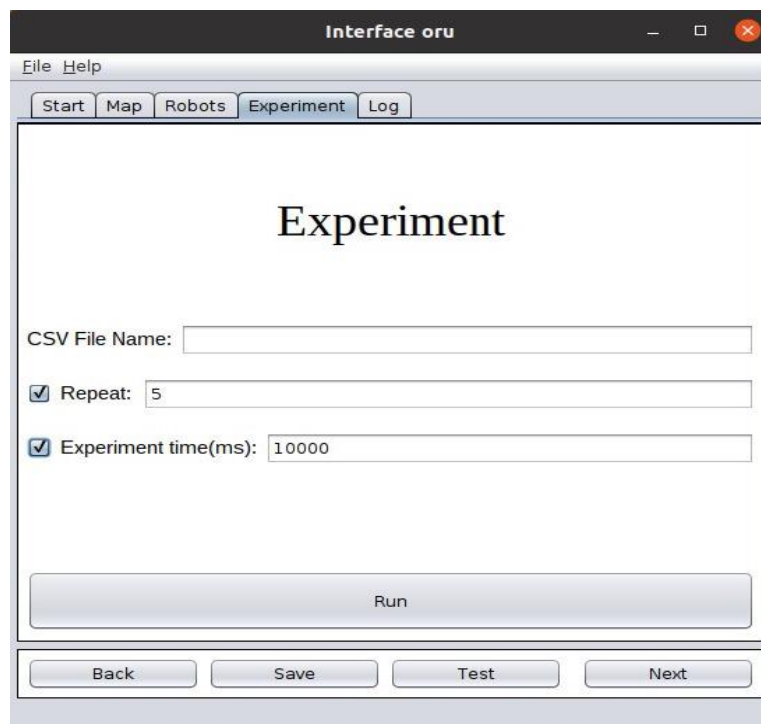
Figur 3.5 : Welcome fönster



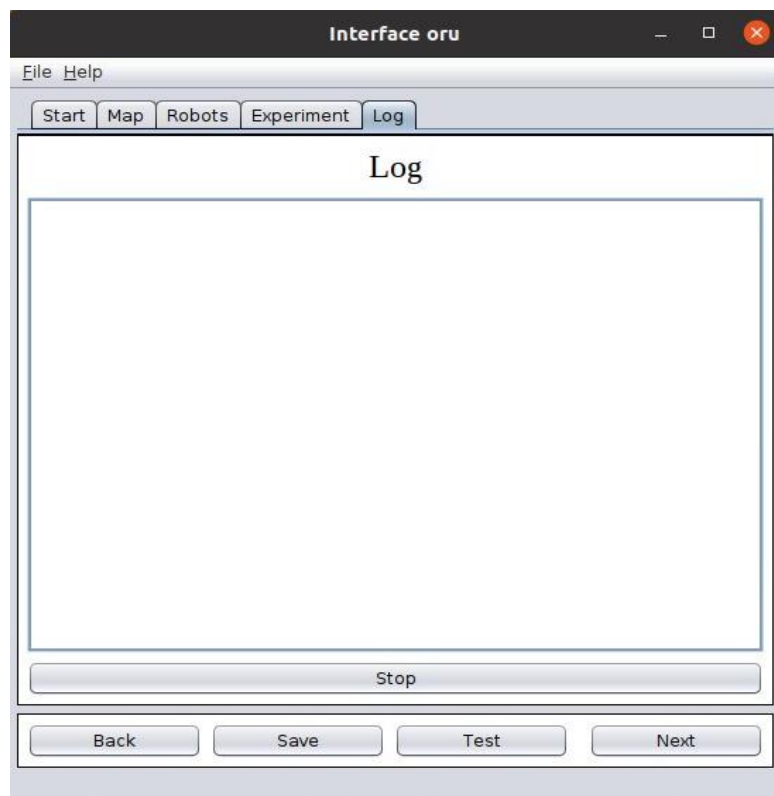
Figur 3.6 : Map fönster



Figur 3.7: Robots fönster



Figur 3.8 : Experiment fönster



Figur 3.9 : Log fönster

Flexible gränssnitt har flera fönster med specifika funktioner. Den är mer utvecklad och har flera komponenter som log, repeat och experiment time. Användare kan följa de fönster Start, Map, Robots, Experiment och Log baserat på den uppgift de vill utföra eller data de vill ändra med möjlighet att spara de valda resultatdata i en CSV-fil. Det guidar användaren bättre eftersom där finns en förslagen "next" för att komma till nästa sida och det är flexibel "back" för att gå tillbaka. Dessa fönster bidrar till att underlätta användarens förståelse och interaktion med systemet genom att tydligt presentera relevanta information och instruktioner vid varje steg. Detta hjälper användaren att navigera smidigt och effektivt genom processen utan att känna sig osäker.

4. Metoder och verktyg

Detta kapitel kommer att beskriva den använda utvecklingsmetod i detta examensarbete samt de verktyg som har använts och har bidragit till arbetets framsteg och resultat.

4.1. Metoder

En agil metod med inkrementell utveckling har använts för detta projekt. Agil metod är tillämplig för att utveckla systemet på ett flexibelt sätt.

I agil metod ingår fem principer [6]:

1. Den första principen handlar om att fokusera på uppdragsgivare som ingår i utvecklingsteam och kan ge feedback på implementering och utvecklingsprocesser. Jag har följt denna princip genom att ha möten med min handledare varje vecka för att få återkoppling på implementeringen och utvecklingsprocesserna
2. Den andra principen handlar om att utvecklare måste ta hänsyn vid designen till ändringar i krav för att uppdragsgivare ska få möjlighet att ändra på kraven under utveckling. Under examensarbetets gång har jag haft diskussioner med min handledare flera gånger, vilket har lett till flera utvecklingar och anpassningar av kraven.
3. Den tredje principen är inkrementell utveckling som handlar om att processen för aktiviteterna kan överlappa varandra och behöver inte ordnas. Till exempel kan utveckling ske under specifikationsprocess, och under utvecklingsprocessen kan utvecklare och/eller uppdragsgivare gå tillbaka till specifikationen och ändra och gå till validering eller tillbaka till utveckling under valideringsprocess. Det har hänt flera gånger, jag har utvecklat under specifikationsprocess, samt under utvecklingsprocessen har jag haft flera krav som jag kunde implementera.
4. Den fjärde principen avser enkelhet, att systemet ska vara enkelt med enkel design vilket ger möjligheter för förbättringar i framtiden. Enkelhet har varit en viktig faktor under min utveckling där fokus har legat på att skapa ren kod och en tydlig design. Genom att följa principen om enkelhet har jag kunnat göra förbättringar och möjliggjort framtida utveckling av systemet på ett effektivt sätt.
5. Den femte principen riktar sig mot individer, stödjer parvis programmering som innebär att programmerare arbetar i par på ett hållbart sätt. Det innebär att de inte behöver arbeta många timmar i sträck vilket leder till

bättre resultat, Den principen passar inte i detta examensarbete eftersom jag har utfört det själv.

Agila principer har använts så mycket som möjligt under denna utveckling för att förbättra kvaliteten och få de önskade resultaten.

4.2. Verktyg

Detta kapitel handlar om olika verktyg som används under utvecklingen. Det finns tre olika typer av verktyg som har använts. Den första typen av verktyg är utvecklingsmiljön (IDL), den andra är ramverket *coordination_oru* och den tredje är dataformatet.

4.2.1. Utvecklingsmiljö(IDL)

- **Visual Studio Code** : en stor del av utvecklingen gjordes i Visual Studio Code, vilket underlättade processen med att skapa och redigera koden. VS ger användare möjligheten att lägga till stöd för olika programmeringsspråk, felsökningsverktyg och olika verktyg för att underlätta utvecklingsarbetet[11].
- **Apache netbeans**: används för att implementera användargränssnitt som ger möjlighet att skapa och anpassa användargränssnitt på ett enkelt och effektivt sätt. NetBeans-plattformen erbjuder en applikations vanliga krav, som till exempel menyer, dokumenthantering och inställningar, utvecklare behöver bara skapa de delar av applikationen som inte redan finns på NetBeans-plattformen. Vid slutet av utvecklingscykeln kan utvecklare sammansätta applikationen tillsammans med NetBeans-plattformen, vilket sparar tid och energi[10].
- **Trello**: är en plattform som används för att planera, organisera och hantera projekt.
- **Creately**: är en plattform som används för att skapa diagram. I detta arbete har komponentdiagram, aktivitetsdiagram och klassdiagram skapats med hjälp av Creately
- **Ubuntu-20.04**: Vid utveckling var det nödvändigt att använda samma operativsystem som *Coordination_oru*-systemet. Därför används Ubuntu 20.04 som operativsystem på datorn.
- **Java** är ett programmeringsspråk som används för att utveckla *Coordination_oru*-systemet, vilket är skrivet i Java-språket och den har använts för utveckling.
- **Github**: används för att dela med mig av koden och kunna få uppdateringar för systemet.
- **Google Drive**: används för att dokumentera och dela min rapport med handledare.

4.2.2. **Coordintion_oru-ramverk :**

Coordination_oru hämtats från Github och de utvecklades av Federico Pecora [9]. Inom *Coordination_oru* har flera klasser utökats för att skapa experimentsystemet. Här är de viktiga klasser som har använts :

1. Missions-klass: Används för att lägga till eller ta bort uppdrag för robotarna.
2. Mission-klass: Används för att skapa ett uppdrag med en start- och slutpunkt, till exempel (x1, y1) -> (x2, y2).
3. Pose-klass: Används för att ange startpositionen för en robot med koordinaterna (x, y, theta).
4. PoseSteering-klass: Används för att lagra och skicka vägpunkter till Mission.
5. Coordinate-klass: Används för att placera punkter som representerar robotens form.
6. ConstantAccelerationForwardModel-klass: En modell där hastighet och acceleration för varje robot kan specificeras.
7. ReedsSheppCarPlanner-klass: Används för att använda en visualisering och placera ut robotarna på den.
8. CriticalSection-klass: Används för att jämföra robotarna när de rör sig och avgöra vilken robot som ska gå först.
9. RobotAtCriticalSection: Används för att kontrollera om det finns kritiska punkter eller inte.
10. RobotReport-klass: Används för att få information om roboten.
11. TrackingCallback-klass: Används för att övervaka och testa robotens position och läge, dvs. används för att känna av robotarnas status.
12. JTSDrawingPanelVisualization-klass: Används för att skapa och konfigurera fönstret för robotarna, det vill säga visualisering.

Flera andra klasser har använts för att testa, utveckla och förstå hur systemet fungerar.

4.2.3. Dataformat

- JSON: JavaScript Object Notation eller JSON är ett dataformat som blir populärt eftersom den kan lätt läsas av både människor och maskiner och används för att överföra data. En JSON-fil innehåller objekt som är samlingar av attribut och dess värden, samt arrayer av värden. JSON stöder flera datatyper som integer, string och boolean. JSON stöds av JavaScript och många andra programmeringsspråk och databashanterare.[\[5\]](#) JSON-formatet har använts för simulering. Det används för att organisera och dela information mellan olika delar simuleringssystemet.[\[2\]](#)
- CSV är ett annat format som ska nyttjats för att spara resultat från simulations experiment som användare behöver för att kunna jämföra och analysera data.

5. Genomförning

I detta kapitel kommer utvecklingen under detta examensarbete att beskrivas. Det kommer att diskuteras vilka data som har specificerats och vilka klasser som har skapats för att utöka ramverket för *Coordination_oru*, samt för att utveckla gränssnittet. Dessutom kommer beskrivningarna av hur experimentspecification sparas i en JSON fil och hur output data från simulation sparas i csv filer.

5.1. Data för specifikation

Ett helt experiment innebär att det finns både indata/input som har specificerats från *Coordination_oru* och utdata/output. Input-data består av en yaml-fil som representerar strukturerad information om kartan, yaml-filen kan enkelt läsas och förstås av *Coordination_oru*-ramverket för att skapa och visa kartan med alla dess egenskaper och element.

För individuellt robot består inputen-data av namn för roboten, hastighet som beskriver hur snabbt eller långsamt en robot rör sig, acceleration som refererar till förändringen av hastigheten över tiden, samt storlek som representerar fyra punkter för koordinatsystemet. Dessutom inkluderas robots färg, robots väg/uppgift som kan anges med start- och slutpunkter eller flera punkter. Dessa data kan antingen presenteras genom att skriva koordinaterna för varje position/punkt, eller genom att klicka och markera punkterna på kartan.

Output-data ger viktig information om experimentets utförande och resultat. Denna data är användbar för att utvärdera experimentet, analysera prestationen hos robotarna och identifiera eventuella problem, datan innehåller följande information:

Antal iterationer: Detta refererar till det totala antalet gånger som experimentet har körts eller uppdaterats.

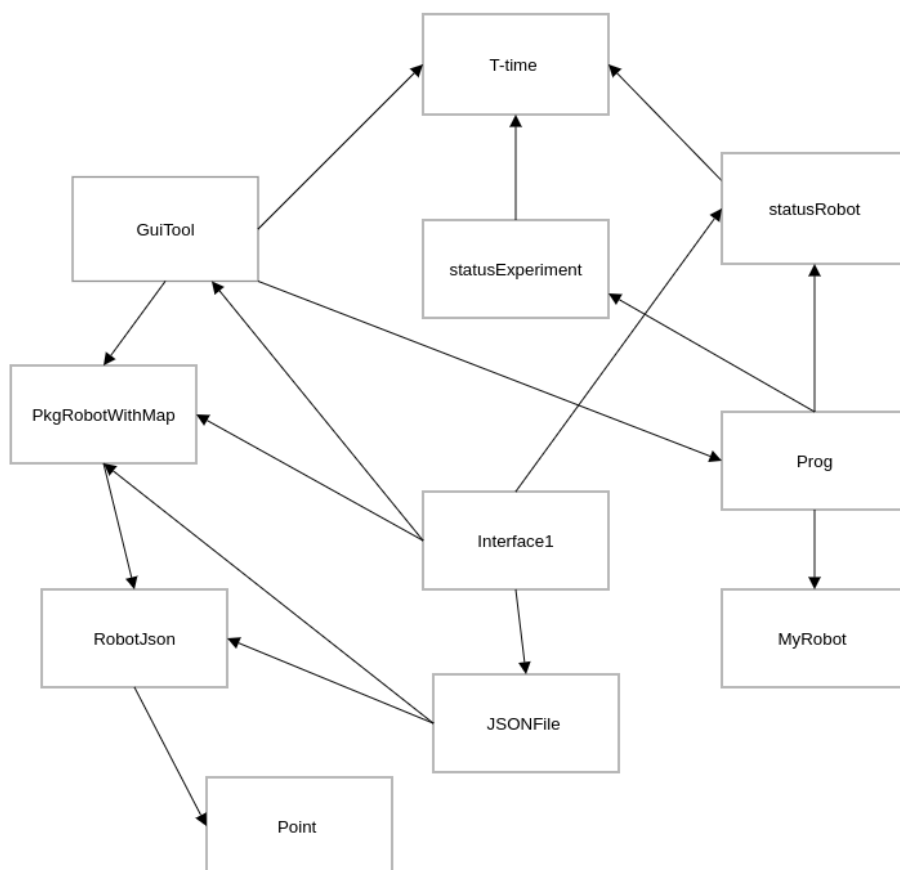
Experimentets sluttid: Detta är tiden då experimentet avslutas eller når sitt mål. Detta hjälper till att mäta den totala tiden som krävdes för att slutföra uppgiften eller för att jämföra olika experiment med avseende på deras sluttider.

Antal konflikter: Detta är det totala antalet konflikter som uppstår mellan robotarna under experimentet.

Robotarnas slut- och väntetid: För varje individuell robot i experimentet inkluderas deras individuella slut- och väntetider. Slutiden är den tidpunkt då roboten slutför sin uppgift eller når målet. Väntetiden är den totala tiden som roboten spenderar i vänteläge under experimentet.

5.2. Experiment klasser

I mappen "gui_oru" återfinns samtliga klasser som har utvecklats för att skapa experimentspecifikationer och utöka *Coordination_oru*. Dessa klasser skapar interaktion med användargränssnittet och hantering av experimentfunktioner.



Figur 5.1 : Enkel klassdiagram

Här finns ett klassdiagram som visar de klasser som har skapats och deras relationer, och nedan följer en beskrivning av vad varje klass gör och varför de har skapats:

- I klassen *JsonFile.java* hanteras JSON-filer genom läsning och skrivning. Denna klass innehåller även verktyg som används i denna process samt funktioner för att skapa en väg mellan två punkter så att roboten kan följa denna väg.

- Klassen RobotJson.java: skapas för att representera varje robot separat. Den innehåller attributen som robotens namn, hastighet, acceleration, färg, storlek/form och väg från en punkt till en annan tills banan är helt klar.
- I klassen GuiTool.java finns verktyg som används i robotsgränssnittet. Denna klass är skapas för att spara information om den nya roboten som har lagts till och/eller ändra robotens data.
- Klassen MyRobot.java innehåller all data och styrning för roboten med följande funktioner:
 - Sätta robotens hastighet.
 - Skapa en uppgift för roboten.
 - Ställa in storlek/form på roboten.
 - Tilldela kartan till roboten.
 - Skapa en färg för roboten.
- StatusRobot.java: skapas för att få information om en robots status och för att lagra robotens status i en CSV-fil.
- Klassen PkgRobotWithMap.java har följande funktioner:
 - Lägg till en robot i listan som innehåller alla robotar.
 - Ta bort en robot från listan.
- Klassen Point.java används för att skapa en punkt med följande attribut: (X, Y, Theta) eller (X, Y). Detta används för att skapa en väg från en punkt till en annan punkt, start - och slutpunkt .
- Klassen interface1.java: Innehåller grafiska gränssnittet för experimentspesifikaton:
 1. I "Welcom" sidan finns två knappar:
 - Load-knapp: för att öppna en JSON-fil
 - Create-knapp: för att skapa en JSON-fil
 2. I "Map" sidan finns en knapp:
 - Open: för att välja en karta och öppna den.
 3. I "Robots" sidan finns:
 - Robots → List of robots, innehåller de befintliga robotar

- New Robot : för att lägga till en ny robot till i listan
- Delete-knapp: för att ta bort en vald robot från listan, på detta sätt roboten tas bort från experimentet som kommer att köras senare.
- Vid modifiering finns det en Apply-knapp: för att tillämpa de ändringar som har gjorts på roboten. När den här knappen trycks in sparas inte robots data i JSON-fil, utan endast för att testa experiment.
- Textfält bredvid Name : för att ange robotens namn eller för att ändra robotens namn.
- Textfält bredvid Velocity : för att ange eller ändra på robotens hastighet.
- Textfält bredvid Acceleration : för att introducera eller modifiera robotens acceleration.
- Textfält bredvid Color : för att ändra på robotens färg
- Textfält bredvid Size : för att lägga till eller ändra på robotens storlek/form.
- Textfält bredvid Path : för att ange vägpunkterna som roboterna kommer att passera genom eller ändra denna väg.
- Add-knapp bredvid path: för att markera start- och slurpunkten i kartan istället för att skriva dem.
- Iteration-knapp: för att ange antal upprepningar för experiment.

4. I "Experiment" sida finns :

- Textfält bredvid CSV fil Name : för att ange namn för CSV-fil.
- Fem olika alternativ att välja mellan för att sparas på CSV-fil.
- Textfält bredvid Repeat : för att ange antal gånger experimentet ska upprepas.
- Textfält bredvid Experiment time : för att ange tiden för alla experimenter.
- Run-knapp för att köra experimentet

5. I "log" sidan finns:

- Det finns en Stop-knapp för att stoppa detta experiment.

6. I alla sidor finns:

- Save-knapp: används för att spara den nya robots data i den öppnade json-fil, det vill säga när vi gör ändringar och vi vill spara dessa ändringar trycker vi på den här knappen.
- Test-knapp: för att testa om det är möjligt.
- Next-knapp: för att gå till nästa sida
- Back-knapp: för att gå tillbaka till förra sidan.

- `StatusExperiment.java`: skapas för att få information om experimentets status och antalet gånger det har körts.
- `T_timer.java`: skapas för att implementera en tidsbaserad timer vid starten av experimentet.
- `Prog.java`: skapas för att samla information om alla robotar, skapa och styra experimentet. Det är klassen som ansvarar för att starta och stoppa experimentet.
- `addPath.ImageWindow.java`: skapas sist för att öppna ett fönster med den karta som används i experimentet och när man klickar på kartan lagras de koordinaterna för vägen till robotens uppdrag.

5.3. JSON-fil:

JSON-format kommer att användas för att spara experimentspecifikation på ett strukturerat sätt. För att systemet kunna hantera en JSON-fil, läsas av eller/och skrivs på, måste JSON-filen ha en specifik struktur. Denna struktur ska omfatta tre objekter:

Objekt "Experiment" har två attribut :

- En sträng som representerar namnet på CSV-fil.
- Ett nummer som anger antal repetitioner för experimentet.

Objekt "Map" har en attribute:

- En sträng som representerar yaml-file som specificerar kartan.

Objekt "Robots" innehåller en list av de specifikationer för individuellt robot och varje robot har sju attribut :

- En sträng som representerar robotens namn.
- Ett nummer som anger robotens hastighet
- Ett nummer som anger robotens acceleration.
- En sträng som anger robotens storlek/form.
- En sträng som anger robotens färg.
- En sträng som representerar robotens start- och slutposition.
- Ett nummer som anger antalet robot-repetitioner i varje experiment iteration.

```

{
  "Experiment": [
    "Maria.csv",
    2
  ],
  "map": "maps\\map-partial-2.yaml",
  "Robots": [
    [
      "MA1",
      10.0,
      4.0,
      "Red",
      "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)",
      "(8.8,30.4,0.0) -> (72.4,9.4,0.0)",
      1
    ],
    [
      "MA2",
      15.0,
      4.0,
      "Blue",
      "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)",
      "(7.5,8.6,0.0) -> (71.1,37.2,0.0)",
      2
    ]
  ]
}

```

Figur 5.2 : Maria.json fil

Den här JSON-filen är ett exempel på hur filen ska se ut. Experiment-objektet refererar till en CSV-fil med namnet "Maria.CSV" och experimenten kommer att upprepas två gånger. Karta-objektet har namnet "maps\\map-partial-2.yaml". Robotobjekten representerar två olika robotar. Den första roboten har namnet "MA1", hastigheten 10, accelerationen 4 och färgen röd. Dess storlek/form är specificerad som "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)", med en startpunkt på (8.8,30.4,0.0) och en slutpunkt på (72.4,9.4,0.0)". Roboten upprepas en gång i varje iteration. Den andra roboten har namnet "MA2", hastigheten 15, accelerationen 4 och färgen blå. Dess storlek/form är specificerad som "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)", med en startpunkt på (7.5,8.6,0.0) och en slutpunkt på (71.1,37.2,0.0)". Roboten upprepas två gånger, vilket betyder att roboten kommer att köra hela vägen två gånger i varje iteration.

5.4. Datainsamling under experiment:

På experimentens fönster erbjuds användaren att ge namn för en CSV-fil och systemet skapar en CSV-fil för att spara de valda data. Det innebär att användaren kan bestämma vilken typ av information som ska finnas i varje kolumn.

Om användaren väljer att spara alla alternativ i en CSV-fil, som i följande exempel (Figur 5.3), leder det till detta resultat: Varje rad innehåller data för ett enskilt experiment, och i exemplet i figur 5.3 visas fyra experiment. Varje kolumn representerar en helt annan typ av information.

- Den första kolumnen visar experimentnumret för den aktuella raden.
- Den andra kolumnen visar sluttid för experimentet.
- Den tredje kolumnen visar antal konflikter som har hänt under experimentet.
- För varje robot finns därefter två kolumner. Den första innehåller sluttid, den andra innehåller väntetid. I figur 5.3 finns kolumner för 2 robotar.

```
1 1,7593 ms,0,R1(Finish): 6415 ms,R1(Wait): 1178 ms,R2(Finish): 7589 ms,R2(Wait): 4 ms,
2 2,16461 ms,0,R1(Finish): 15767 ms,R1(Wait): 694 ms,R2(Finish): 16456 ms,R2(Wait): 5 ms,
3 3,25235 ms,0,R1(Finish): 24504 ms,R1(Wait): 731 ms,R2(Finish): 25232 ms,R2(Wait): 3 ms,
4 4,34245 ms,0,R1(Finish): 33034 ms,R1(Wait): 1211 ms,R2(Finish): 34241 ms,R2(Wait): 4 ms,
```

Figur 5.3 : Test.csv

6. Resultat

I detta kapitel presenteras resultatet i form av användargränssnittet för systematiskt experimenterande att presenteras. De olika komponenterna i det koordinerade systemet kartor, vägar, uppdrag, robotar, experiment och andra element har skapades möjlighet för att specificeras och implementeras för att skapa ett komplett experiment. *Coordination_oru*-ramverket har utökats för att automatisera experiment baserat på specifikationen. Därefter har ett lämpligt användargränssnitt implementerats. Dessutom samlas valda data in och rapporteras i en CSV-fil som beskriver systemets resultat.

I varje fönster presenteras huvudsektionerna "Fil" och "Hjälp". I sidfoten finns knapparna "Back", "Save", "Test" och "Next", vilka finns på samtliga gränssnitt.

"Save"-knappen används för att spara den nuvarande specifikationen i JSON fil.

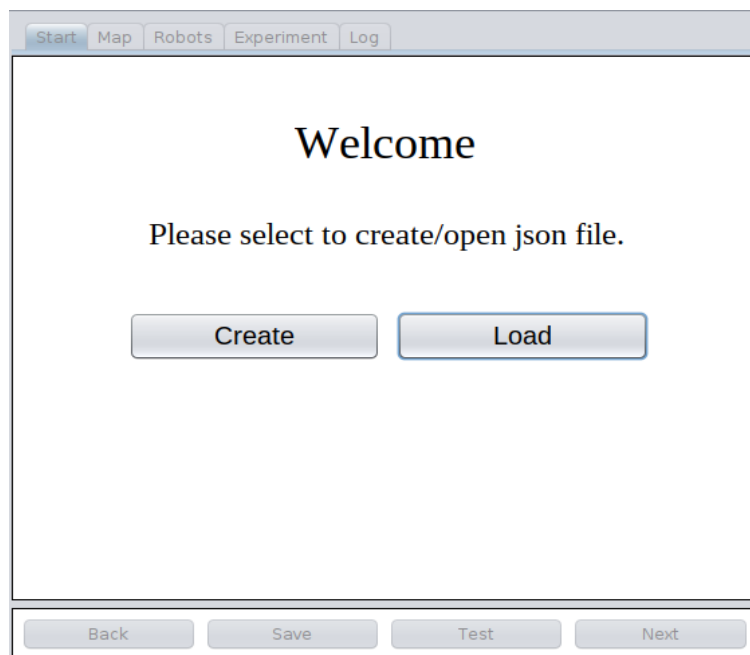
"Test"-knappen används för att testa experimentet.

"Next"-knappen används för att navigera till nästa sida.

"Back"-knappen används för att gå tillbaka till föregående sida.

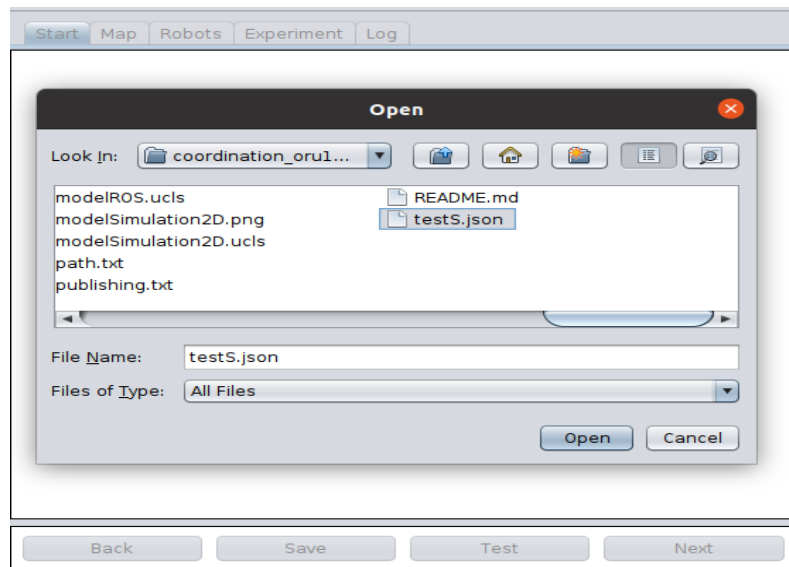
På startsidan får användaren två alternativ: öppna en befintlig JSON-fil eller att skapa en ny. I följande avsnitt kommer dessa två alternativ att presenteras i detalj.

6.1. Öppna en JSON-fil:



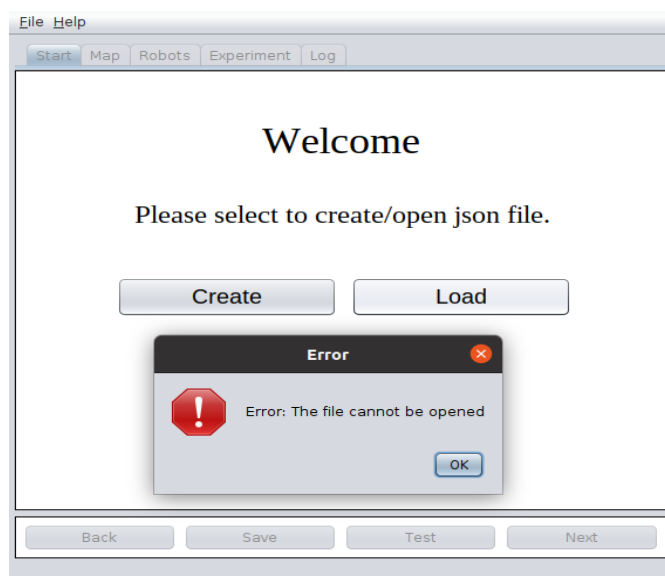
Figur 6.1: Första sidan

När användaren väljer alternativet "Load" för att öppna en befintlig JSON-fil, öppnas ett fönster där användare kan ange sökväg till den JSON-filn.



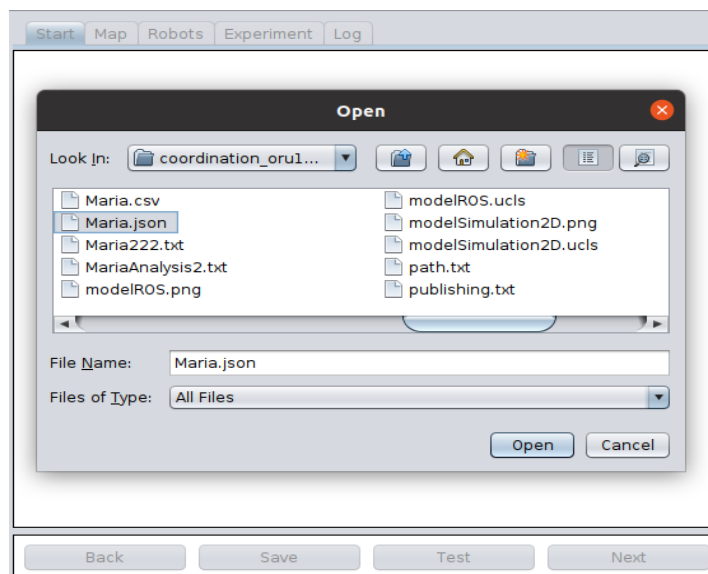
Figur 6.2 : öppna fil

Efter att en JSON-fil har valts kan användaren trycka på "Open" för att öppna filen. Om JSON-filen är korrekt och följer rätt struktur kommer datan att läsas in och visas men om filen inte är korrekt strukturerad kommer den inte att öppnas och användare kommer att få ett meddelande att filen kan inte öppnas. I detta exempel har en JSON-fil med namnet "testS.Json" valts, och eftersom den inte följer strukturen kan den inte öppnas och detta meddelande visas i figur 6.3.

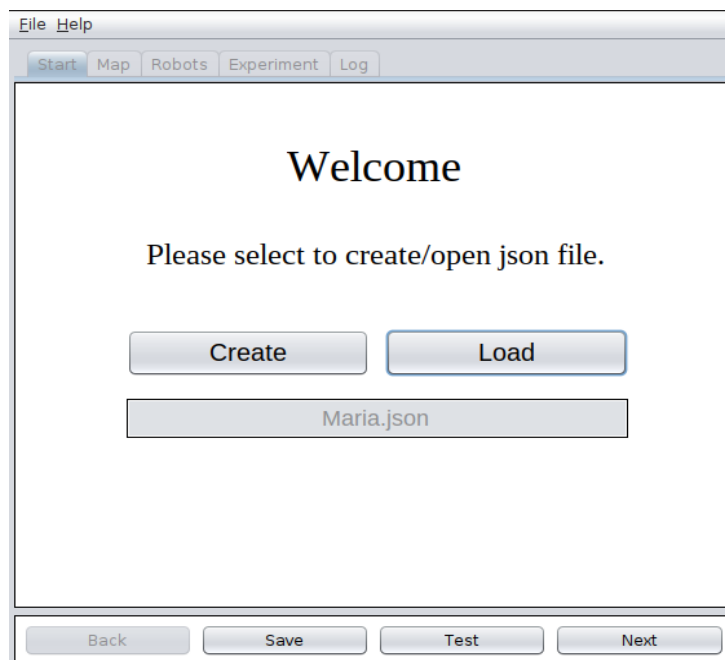


Figur 6.3: felaktig fil

En annan fil som heter "Maria.json" har valts och eftersom den följer rätt struktur kan den öppnas och resultatet visas i figur 6.5.



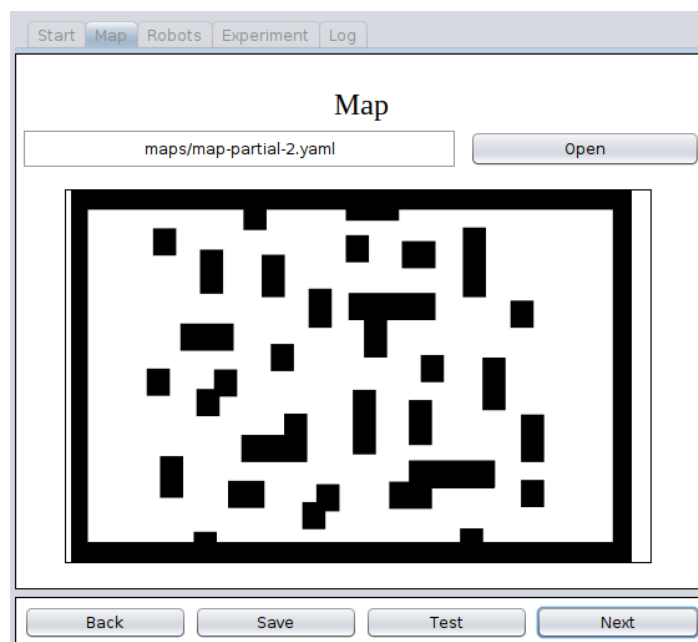
Figur 6.4: korrekt fil



Figur 6.5: Maria.json öppnas

Genom att klicka på knappen "Next" får användare fortsätta till nästa sida.

6.1.1. Map sidan



Figur 6.6: Aktuella kartan

På "Map" sidan, som visas i figur 6.6, kan användaren se kartan som filnamn angavs och är specificerats i valda JSON-filen, "Maria.json", och har möjlighet att byta till en annan karta genom att använda knappen "Open". Om användaren väljer att öppna en annan karta kommer den nya kartan att visas direkt. Yaml är ett dataformat som används för att representera strukturerad information om kartan.

Genom att klicka på knappen "Next" får användare fortsätta till Robots sida.

6.1.2. Robots sidan

Interface oru

File Help

Start Map Robots Experiment Log

Robots

R1
R2

Name: MA1

Velocity: 15.0

Acceleration: 4.0

Color: Red

Size: (-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)

Path: (5.1,40.6,0.0) -> (71.1,7.7,0.0)

Iteration: 3

New Robot Delete Apply

Back Save Test Next

Figur 6.7: Robot 1

Interface oru

File Help

Start Map Robots Experiment Log

Robots

R1
R2

Name: MA2

Velocity: 15.0

Acceleration: 4.0

Color: Blue

Size: (-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)

Path: (7.5,8.6,0.0) -> (71.1,37.2,0.0)

Iteration: 4

New Robot Delete Apply

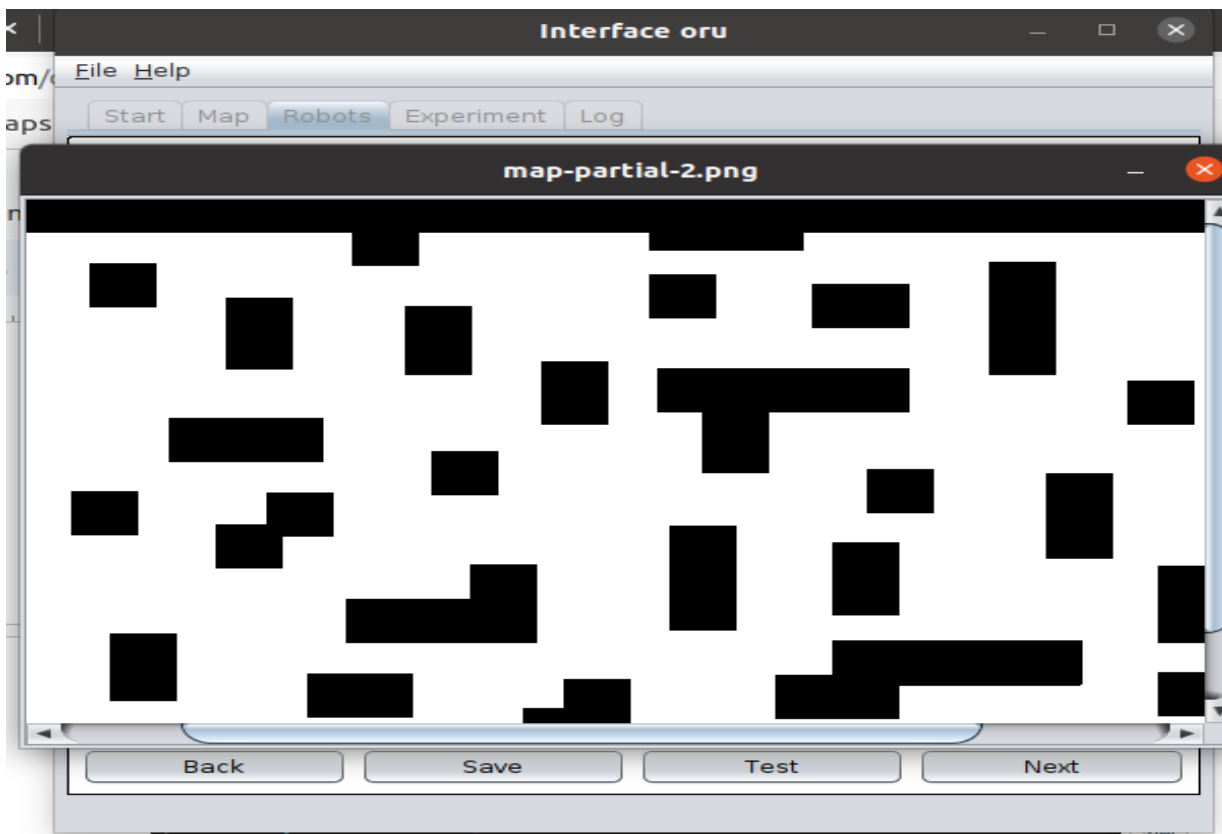
Back Save Test Next

Figur 6.8: Robot 2

På "Robots" sidan, som visas i figur 6.7 och figur 6.8, kan användaren se information om robotarna som finns i den valda JSON-filen. Figur 6.7 visar att den första roboten har namnet "MA1", hastigheten 15, accelerationen 4 och färgen röd. Dess storlek/form är specificerad som "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)", med en startpunkt på (5.1,40.6,0.0) och en slutpunkt på (71.1,7.7,0.0)". Roboten upprepas tre gånger i varje iteration.

Figur 6.8 visar att den andra roboten har namnet "MA2", hastigheten 15, accelerationen 4 och färgen blå. Dess storlek/form är specificerad som "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)", med en startpunkt på (7.5,8.6,0.0) och en slutpunkt på (71.1,37.2,0.0)". Roboten upprepas fyra gånger i varje iteration.

Om användaren inte kan skriva siffror för positioner, finns möjlighet att välja den i kartan som visar figur 6.9.



Figur 6.9: Start- och slutpunkt/ vägpunkter

Knappen "Add" kan användas för att ange startpunkt, mellanpunkter och slutpunkt genom att klicka direkt på kartan istället för att skriva koordinater för dessa punkter. På så sätt kan användaren enkelt markera önskade platser på kartan som visar figur 6.9 för robotens rörelse.

Användare kan lägga till en ny robot till i listan genom att använda knappen "New Robot" samt att ange information såsom robots-hastighet, acceleration, färg, storlek/form, startpunkt, slutpunkt och upprepning.

Användare kan ta bort en vald robot från listan genom att använda knappen "Delete", på detta sätt roboten att tas bort från listan.

Robotarnas hastighet kan ställas in för varje individ. Det är viktigt att välja lämpliga hastigheter för varje robot för att uppnå en effektiv samordning. För låga hastigheter kan leda till onödig fördröjning medan för höga hastigheter kan öka risken för kollisioner och begränsa robotarnas förmåga att samarbeta. Acceleration är en viktig faktor eftersom den påverkar robotarnas förmåga att effektivt navigera och utföra sina uppgifter. Om robotarna saknar tillräcklig acceleration kan de möta svårigheter när de behöver ändra riktning eller hastighet för att undvika hinder eller samarbeta med andra robotar.

Användaren kan ändra robotarnas data genom att ändra värdena i de textfält som är tillgängliga. Vid ändringar bör användare använda knappen "Apply" för att spara ändringar i detta experiment.

För att spara ändringar i JSON-filen bör användare använda knapp "Save". När den här knappen trycks inte sparas inte robots data i JSON-fil, utan endast testa experiment.

6.1.3. Experiment sidan

Efter att användaren har avslutat specification av robotar och klickat på "next" kommandet användaren till "Experiment" sidan som visas i figur 6.10

Interface oru

File Help

Start Map Robots Experiment Log

Experiment

CSV File Name: Maria1.csv

☒ Iteration of experiment ☒ End time of experiment ☒ Number of collisions

☒ End time of robots ☒ Waiting time for each robot

☒ Repeat: 4

☐ Experiment time(ms):

Run

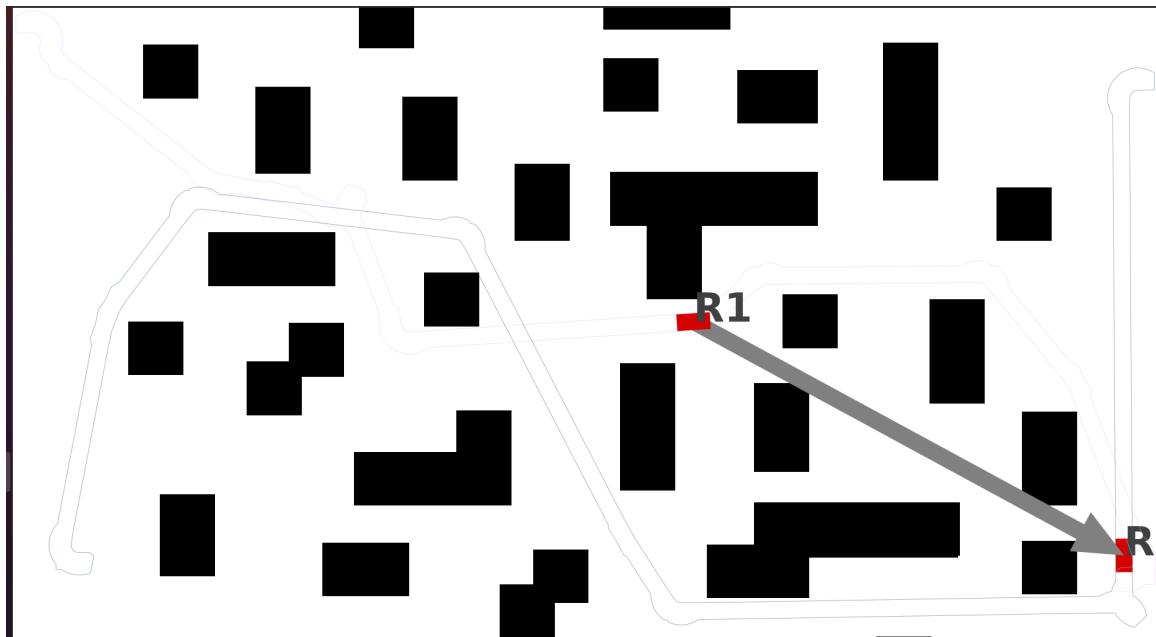
Back Save Test Next

Figur 6.10: Experiment

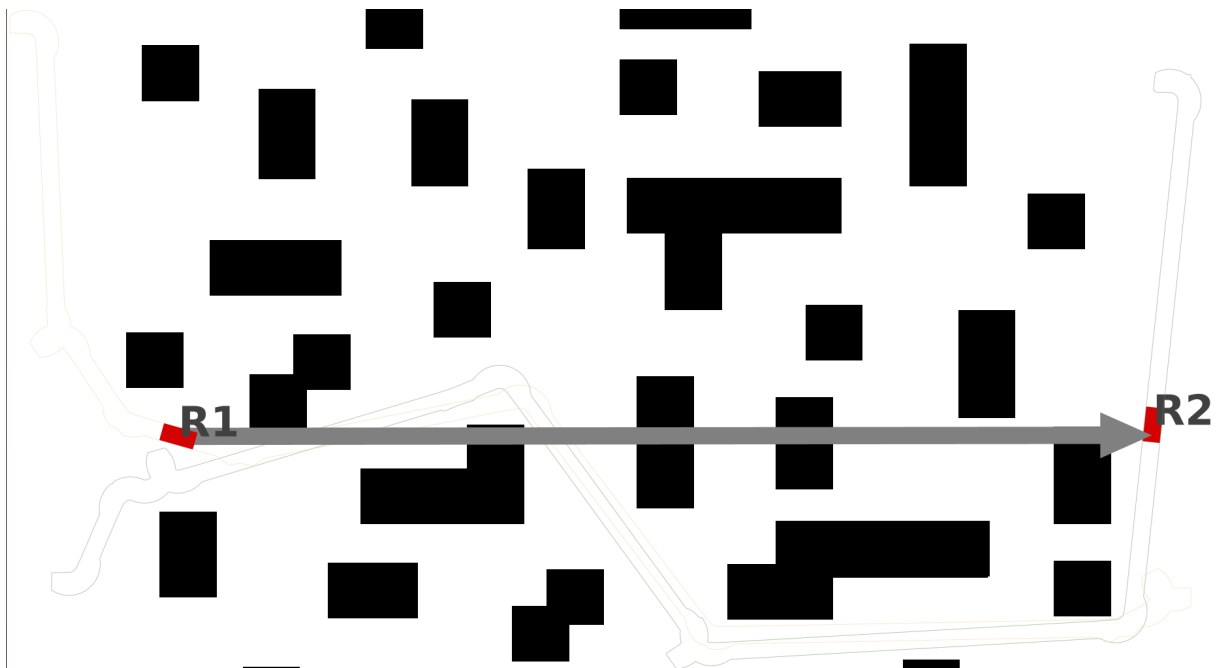
Användaren kan ange ett namn för CSV-filen och välja bland de fem olika alternativen för att bestämma vilken typ av data som ska sparas i CSV-filen, iteration nummer, slutid för varje robot i detta iteration, antal kollisioner som här hänt under detta iteration och/eller sluttid för hela simulationkörning. Dessutom kan användaren ange antalet gånger experimentet ska upprepas och ange en tidsgräns för när experimentet ska avslutas.

6.1.4. Användargränssnitt under experiment/test

Dett finns två möjligheter att observera dynamiken. Standard visualisering av *Coordination_oru* och log.



Figur 6.11: visualisering 1



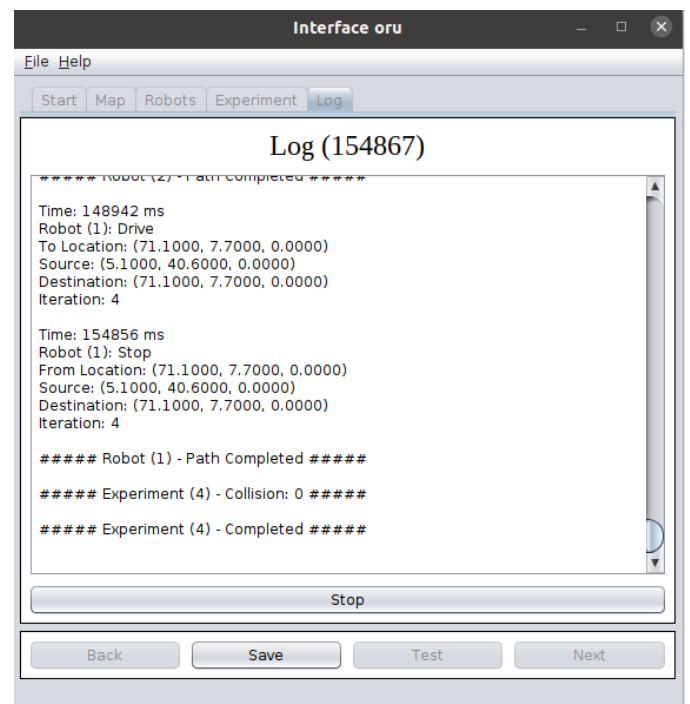
Figur 6.12: visualisering 2

När användaren klickar på "Run" knappen körs experimentet och användaren får en direkt visualisering av hur robotarna rör sig med den angivna datan och experimentinställningarna. Visualisering visas i figur 6.11 och figur 6.12. Det ger användaren möjlighet att analysera robotarnas beteende och resultat i realtid. Då blir det lättare för användaren att förstå och bedöma resultatet av experimentet.

På "Log" sidan visas all information och data om robotarnas och experimentets status över tiden.



Figur 6.13 : Log1



Figur 6.14: Log2

Här får användaren tillgång till all information om hur robotarna rör sig och hur experimentet fortskrider. Informationen som visas under tiden inkluderar tidpunkt, robotens läge, start- och slutdestination, källa, destination och iterationsnummer. I figur 6.13 visas loggen från tidpunkten 0 ms och i figur 6.14 visas loggen när experimentet avslutas efter 154867 ms.

```

1 1,37856 ms,0,R1(Finish): 37855 ms,R1(Wait): 1 ms,R2(Finish): 30622 ms,R2(Wait): 7234 ms,
2 2,76371 ms,0,R1(Finish): 76367 ms,R1(Wait): 4 ms,R2(Finish): 69241 ms,R2(Wait): 7130 ms,
3 3,115799 ms,0,R1(Finish): 115795 ms,R1(Wait): 4 ms,R2(Finish): 108616 ms,R2(Wait): 7183 ms,
4 4,154865 ms,0,R1(Finish): 154856 ms,R1(Wait): 9 ms,R2(Finish): 147456 ms,R2(Wait): 7409 ms,

```

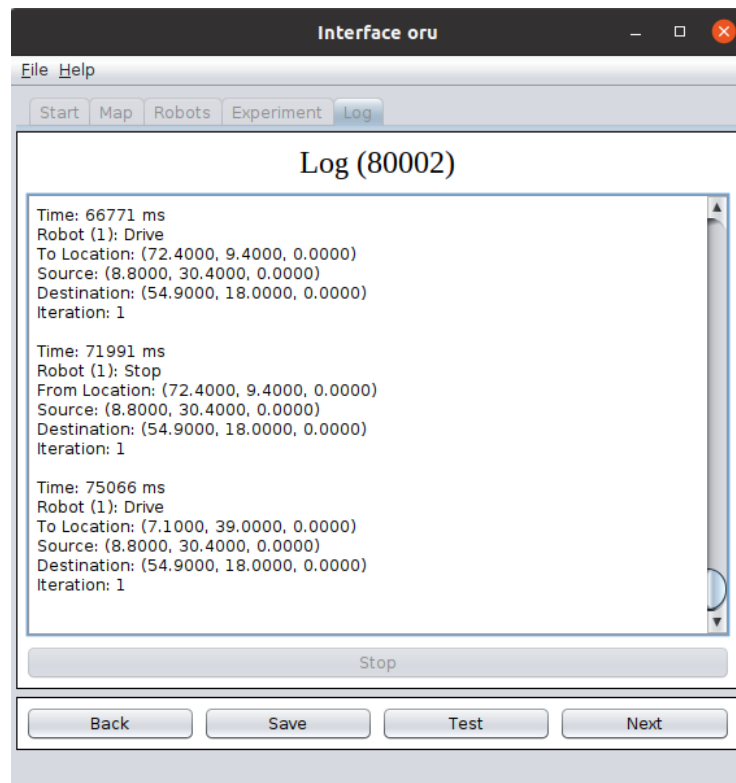
Figur 6.15: Maria1.csv

En CSV-fil med namn Maria1.csv som har välts i figur 6.10 kommer att genereras automatiskt och den valda datan kommer att sparas i filen.

Figur 6.15 visar data för detta experiment, förklaring till varje rad och kolumn i filen finns i kapitel 5, avsnitt 4.

Om användare ger en experiment en tidsbegränsning till 80000 ms:

Figur 6.16: Begränsad tid



Figur 6.17: Log med begränsning

Som i figur 6.16 kommer systemet att stoppas efter 80000 ms även om experimentet inte är klart. Systemet stoppas efter 80002 ms, resultatet visas i figur 6.17.

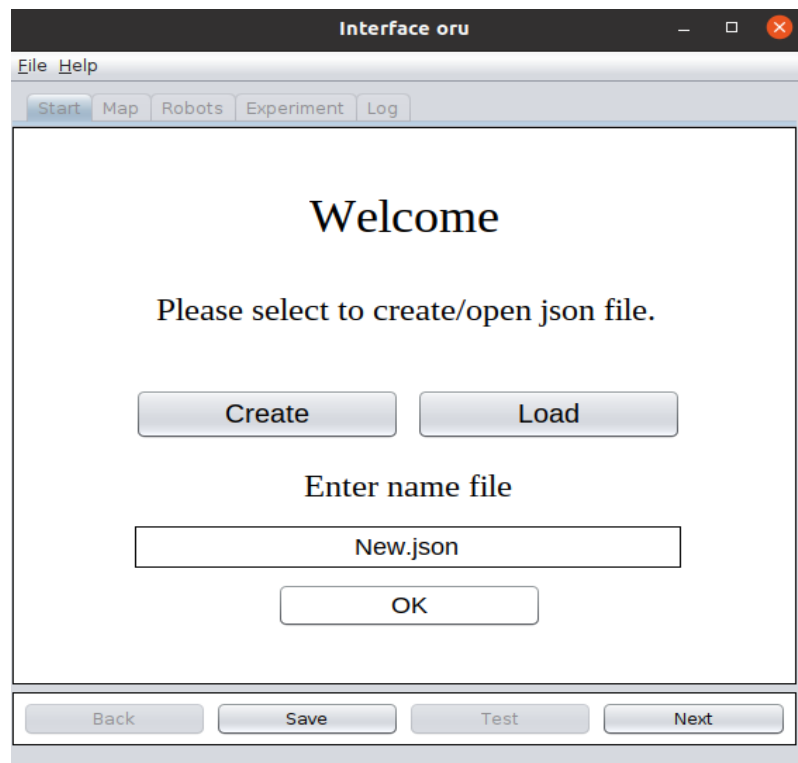
```
{ } Maria.json ●
home > maria6789 > Downloads > coordination_oru1new-Maria-finale (copy) > { } Maria.json > ...
1 {
2   "Experiment": ["Maria1.csv", 4],
3   "map": "maps/map-partial-2.yaml",
4   "Robots": [
5     [
6       "MA1",
7       15.0,
8       4.0,
9       "Red",
10      "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)",
11      "(5.1,40.6,0.0) -> (71.1,7.7,0.0)",
12      4
13    ],
14    [
15      "MA2",
16      15.0,
17      4.0,
18      "Blue",
19      "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)",
20      "(7.5,8.6,0.0) -> (71.1,37.2,0.0)",
21      4
22    ]
23  ]
24 }
```

Figur 6.18 : visar utseendet på den befintliga Maria.json-fil

```
{ } testS.json ×
home > maria6789 > Downloads > coordination_oru1new-Maria-finale (copy) > { } testS.json > ...
1 {
2   "Experiment": ["Maria1.csv"],
3   "map": "maps/map-partial-2.yaml",
4   "Robots": [
5     ["MA1", 4],
6     [
7       "MA2",
8       15.0,
9       4.0,
10      "Blue",
11      "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)",
12      "(7.5,8.6,0.0) -> (71.1,37.2,0.0)",
13      4
14    ]
15  ]
16 }
```

Figur 6.19 : visar utseendet på den befintliga testS.json-fil

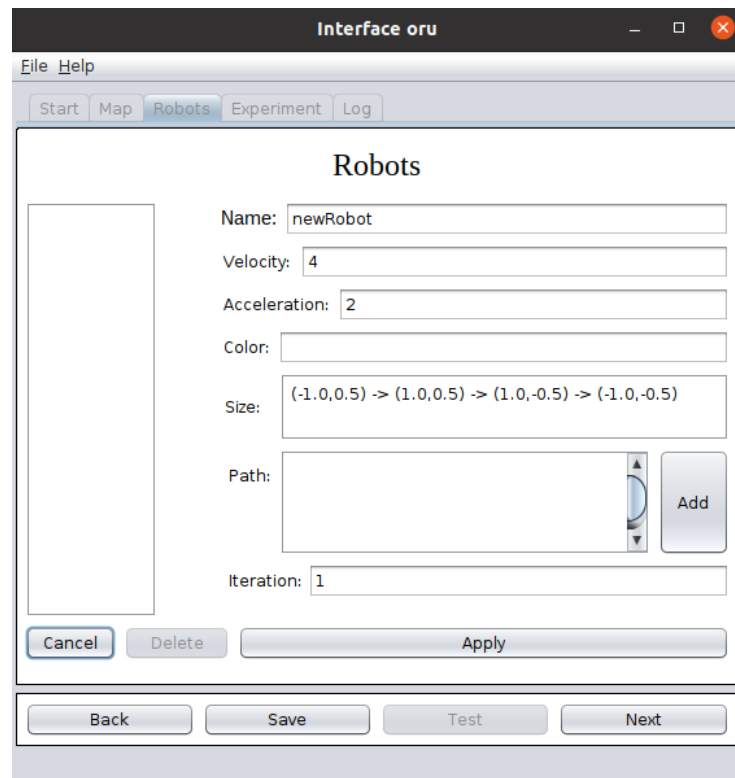
6.2. Process om man börja med ny JSON fil:



Figur 6.20: skapa en New.json

När användaren väljer alternativet "Create" får användaren möjligheten att skapa en ny JSON-fil och ange ett filnamn, som visas i figur 6.20.

På "Map" sidan måste användaren välja en karta för att kunna fortsätta till nästa steg. På "Robots" sida måste användaren skapa minst en robot för att kunna fortsätta till nästa steg.



Figur 6.21 Default

Genom att använda knappen "New Robot" får användaren en ny robot med vissa förinställda värden som kan ändras. Robotsnamnet sätts som "newRobot", hastigheten sätts till 4, accelerationen sätts till 2, och storlek/form sätts till "(-1.0,0.5) -> (1.0,0.5) -> (1.0,-0.5) -> (-1.0,-0.5)". Färgen och start- samt slutpunkten måste anges av användaren.

På "Experiment" sidan för användare samma möjligheter att ange ett namn för CSV-filen och välja bland de fem data som ska sparas i CSV-filen. Och för två möjligheter att observera dynamiken. Standard visualisering av *Coordination_oru* och log.

JSON-fil kommer att skapas direkt och data kommer att sparas i filen när användare klickar på knappen "Save".

7. Diskussion

Detta kapitel presenterar diskussionen om de uppnådda resultaten.

7.1. Uppnående av projektmål:

Genom att titta tillbaka på projektmål som specificeras för systemet i kapitel 1, avsnitt 3, uppstår följande mål:

- Identifiera och analysera de beståndsdelar som ingår i det koordinerade systemet som behöver specificeras:

I arbetet har framgångsrik identifiering och analys utförts av delarna som kartor, vägar, start- och slutpunkt, robotens hastighet, acceleration, färg och andra komponenter. En grundlig analys har genomförts för att förstå deras relationer och interaktioner inom systemet.

- Skapa en experimentspecifikation med alla delar som behövs för att fullt konfigurera systematiska experiment:

Experimentspecifikation har skapats och kan sparas i JSON-format som innehåller alla nödvändiga delar för att konfigurera systematiska experiment. Specifikationen innehåller allt som behövs för att göra experimentet framgångsrikt.

- Utöka *Coordination_oru* för att automatisera experimentation baserad på experimentspecifikation:

Flera klasser och funktioner har skapats för att utöka *Coordination_oru* och möjliggöra en effektiv och smidig hantering av experimentdata och styrning av systemet.

- Implementera ett lämpligt användargränssnitt:

Ett användargränssnitt med olika fönster har implementerats som erbjuder en effektiv metod för användaren att ange och konfigurera experimentspecifikationen, och det ger också en tydlig visuell representation av systemets tillstånd och resultat.

- Samla in vald data i en CSV-fil:

Data har samlats in och sparats i en CSV-fil på ett framgångsrikt sätt.

7.2. Gränssnitt-principer

En betydande del av projektet har fokuserat på att skapa ett användargränssnitt. Under utvecklingen har jag särskilt beaktat flera principer för att uppnå detta.

Flexibilitetsprincipen har varit prioriterad, med ett gränssnitt som kan möta deras specifika behov som att användare kan välja positioner för robot i kartan som startpunkt och slutpunkt, om användare inte kan skriva koordinater för dessa punkter. Användare har möjligheten att testa alla ändringar innan de bekräftas och sparas.

Tydlighet och synlighet av funktioner och information har varit viktigt, med tydligt synliga funktioner och information för att användarna enkelt ska kunna navigera och interagera med gränssnittet. Konsistens i gränssnittet har också varit en viktig princip.

För att underlätta kommunikationen mellan användare och system har olika sätt att interagera med gränssnittet erbjudits som att användare kan skapa en JSON-fil eller öppna en befintlig JSON-fil, med tydlig återkoppling på vad som händer i bakgrunden under användning.

Snabb och smidig kommunikation mellan systemet och användaren har varit prioriterad, med responsiv och effektiv interaktion. Användare ser det när de ändrar på data och testar, vilket ger en snabb och smidig kommunikation

Slutligen jag har fokuserat på att användarna får de verktyg som behövs för att kunna slutföra sina uppgifter på bästa sätt.

7.3. Påverkan på samhället

Coordination_oru är ett betydelsefullt ramverk för automatiserade lösningar som inkluderar multi-robotar, och mitt examensarbete syftar till att göra detta system mer tillgängligt för användare.

En av de viktiga fördelarna med mitt examensarbete är att användaren inte behöver vara bekant med Java-språk för att kunna köra ett experiment i *Coordination_oru*. Istället är det viktigare att användaren har kunskap om specifikationen för experimentet. Det betyder att användaren kan genomföra hela experimentet, samla in resultat och analysera dessa resultat även om de inte har kunskaper i Java eller är Kunniga programmerare.

Denna tillgänglighet och användarvänlighet gör *Coordination_oru* till ett ramverk som kan användas av en bredare målgrupp av användare, oavsett deras tekniska kunskaper eller programmeringsbakgrund. Användare som inte är programmerare

kan dra nytta av systemet och genomföra experiment utan att behöva bekymra sig om att lära sig ett nytt programmeringsspråk.

Detta möjliggör en mer inkluderande och flexibel användning av *Coordination_oru* och öppnar upp stora möjligheter för forskare, ingenjörer och andra användare som är intresserade av att utforska och utveckla multi-robotlösningar.

7.4. Framtid arbete

I framtida arbete kommer fokus att ligga på att fortsätta förbättra och utveckla systemet. En viktig aspekt kommer att vara att erbjuda användare en hjälpfunktion där de kan få information om hur systemet fungerar genom att trycka på "Hjälp"-knappen. Där kommer användare att få information om hur systemet fungerar, information om eventuella problem som kan uppstå under användning och även tillhandahålla lösningar för dessa problem.

En annan viktig punkt är att möjliggöra för användare att ändra färgen på robotarna i visualiseringsfönstret istället för att bara spara färgen som egenskap för roboten. Det skulle göra det möjligt att se olika färger på robotarna i visualiseringen istället för att de alla är röda.

En annan idé som kan vara bra att utforska är möjligheten för användare att ange robotens storlek genom att rita dess form istället för att ange koordinater för robotens form. Detta kan göra det enklare för användare att skapa önskad robotdesign.

Att göra användartester är en viktig punkt för att se hur systemet fungerar i verkligheten. Genom att låta användarna prova systemet kan man få viktig information och återkoppling för att göra det bättre och mer användbart.

För närvarande används en standardiserad rörelseplanering som är fördefinierad i *Coordination_oru*. Det kan vara bra att ge användaren möjlighet att välja mellan olika rörelseplanering algoritmer för varje robot. Detta skulle ge användaren möjlighet att anpassa och välja en lämplig rörelseplanering som passar specifika behov och situationer. Det kan bidra till att förbättra systemets prestanda och möjliggöra användning av mer avancerade algoritmer som är mer anpassade för robotarnas beteende och omgivningen.

Dessa förbättringar kommer att skapa en bättre användarupplevelse och underlätta för användare att använda systemet på ett mer effektivt sätt.

8. Referenser

- [1] F. Pecora, H. Andreasson, M. Mansouri, & V. Petkov. (2018). A Loosely-Coupled Approach for Multi-Robot Coordination, Motion Planning and Control. *Proceedings of the International Conference on Automated Planning and Scheduling*, 28, pp.485-493.
<https://ojs.aaai.org/index.php/ICAPS/article/view/13923/13772>
- [2] P. Wilsdorf, M. Dombrowsky, A. M. Uhrmacher, J. Zimmermann & U. v. Rienen. (2019). Simulation Experiment Schemas Beyond Tools and Simulation Approaches. *Winter Simulation Conference (WSC)*, National Harbor, MD, USA, pp. 2783-2794. doi: 10.1109/WSC40007.2019.9004710.
- [3] P. Wilsdorf , A. Wolpers, J. Hilton, F. Haack & A. M. Uhrmacher. (2023). Automatic Reuse, Adaption, and Execution of Simulation Experiments via Provenance Patterns. *ACM Transactions on Modeling and Computer Simulation*. Volume 33, Issue 1-2, Article No.: 4, pp 1–27, <https://doi.org/10.1145/3564928>
- [4] P. Davidsson, F. Klügl, & H. Verhagen. (2017). Simulation of Complex Systems. In *Springer Handbook of Model-Based Science* (1st ed., pp. 783–797). https://doi.org/10.1007/978-3-319-30526-4_35
- [5] P. Bourhis, J. L. Reutter, F. Suárez, and D. Vrgoč. (2017). JSON: Data model, Query languages and Schema specification. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '17)*. Association for Computing Machinery, New York, NY, USA, 123–135. <https://doi.org/10.1145/3034786.3056120>
- [6] L. Sommerville. *Software Engineering Tenth Edition*. (2004). Kap.3 – Agile software development. S.76. Pearson Education Limited. Edinburgh Gate, Harlow, Essex CM20 2JE, England.
<https://docs.google.com/viewer?a=v&pid=sites&srcid=dWhkLmVkdS5pcXxob2dlci1tYWhtdWR8Z3g6NTYwZWVjNWM3NzdjMDJIZA>
- [7] F. Pecora. The coordination_oru Java Library. A Framework for Multi-Robot Motion Planning, Coordination and Control. Center for Applied Autonomous Sensor Systems, Örebro University.
[Coordination_oru-description.pdf \(ai4europe.eu\)](https://ai4europe.eu/Coordination_oru-description.pdf)
- [8] A. Dix, J. Finlay, G. D. Abowd, R. Beale. (2004). *Human-Computer Interaction*, Third edition, p: 259-288.
- [9] https://github.com/FedericoPecora/coordination_oru (hämtat: 2023-03-01)

- [10] A. Stashkova, C. Pickersgill. (2016). NetBeans Developing Applications with NetBeans IDE, Release 8.1, E61618-03. Kap 5.
<https://docs.oracle.com/netbeans/nb81/netbeans/NBDAG.pdf>
- [11] J. K. Rask, F. P. Madsen, N. Battle, H. D. Macedo & P. G. Larsen. (2020). Visual Studio Code VDM Support. *Aarhus University, Department of Engineering*.
https://www.researchgate.net/publication/346680627_Visual_Studio_Code_VDM_Support