

Anker Engelundsvej 1, 101A ■ 2800 Kgs. Lyngby

TITEL: Map Making
KURSUS: 31380 - Intelligente systemer
KURSUSPERIODE: 13 ugers kursus efteråret 2006

DELTAGERE:

Rasmus Buchwald (s974099)

Jørn Villesen Christensen (s031877)

Thomas Kleis (s042447)

VEJLEDERE:
Morten Lind

OPLAGSTAL: 1
SIDEANTAL: 51
APPENDIKS: 3
AFSLUTTET: 22. december 2006

Synopsis

Denne rapport dokumenterer et program, skrevet i Jess, som får en SMR (Small Mobile Robot) til at kortlægge sit omgivende miljø. Miljøet består af et vejnet i form af en sort linie på et gulv. SMR'en er i stand til at detektere sideveje, vejkryds samt blinde veje, og forbinde disse med hinanden, således at der ud fra facts i en database kan beskrives et kort af vejnettet. Detekteringen og udvælgelsen af hvilke vejkryds der skal udforskes, gøres på en intelligent måde, således at SMR'en ikke blot kører rundt og tilfældigt kortlægger miljøet. Kortet kan ydermere vises i et grafisk vindue, som opdateres løbende efterhånden som SMR'en kommer frem. I dette vindue fremgår det desuden hvor stor en fejl hvert vejkryds er behæftet med, samt hvor mange gange det er besøgt.

Indhold

1	Introduktion	4
2	Analyse	6
2.1	Realtidsproblematikken	7
2.2	Grundlæggende kørsel	7
2.3	Strukturering af data	8
2.4	Udforskning af vejnettet	9
2.5	Håndtering af odometrifejl	11
2.6	Grafisk repræsentation	12
2.7	Kørsel med flere SMR'er	12
3	Implementering	14
3.1	Realtidsproblematikken	14
3.2	Strukturering af data	15
3.3	Grundlæggende kørsel	18
3.4	Udforskning af vejnettet	19
3.5	Grafisk repræsentation	21
3.6	Kørsel med flere SMR'er	22
4	Resultater	24
4.1	Kørsel med én SMR	24
4.2	Kørsel med flere SMR'er	27

5	Forbedringer	31
6	Konklusion	32
A	Opstart	33
B	Templates	34
B.1	Verticelist (Liste med alle vejkryds)	34
B.2	SmrVar (Hjælpevariable)	34
B.3	Vertice (Vejkryds)	35
B.4	LineEnd (Sidevej)	35
B.5	Line (Vej)	36
C	Jess-kode	37
C.1	robots.jss	37
C.2	go.jss	37
C.3	templates.jss	38
C.4	SmrComm.jss	39
C.5	MapManager.jss	45
C.6	gui.jss	47

Til den overordnede styring af SMR'en samt kortlægningen, er sproget Jess benyttet. Jess forbinder til SMRDemo, og kan dermed agere 'Planlægger'.

Det antages, at læseren har kendskab til Jess, SMRDemo og Visualizer.

Analyse 2

Projektet i sin helhed indeholder en række problemstillinger, som i dette afsnit vil blive undersøgt og analyseret nærmere. Disse problemstillinger er:

Realtidsproblematikken Der skal opbygges en metode hvorpå SMRDemo kan håndtere den tidskritiske styring af selve robotten, mens Jess kan tage sig af den strategiske (ikke-tidskritiske) styring. Dvs. SMRDemo skal sikre at SMR'en følger linien og stopper ved vejkryds og blinde veje. Jess skal så blot fortælle hvor SMR'en skal hen. Hvordan sikres synkronisering ml. Jess og SMRDemo?

Grundlæggende kørsel Hvordan undersøges et vejkryds? SMR'en skal være i stand til at detektere vejkryds og hvilke sideveje der er fra krydset. Hvordan sikres at krydset opfattes ens vilkårligt hvor SMR'en kommer fra?

Strukturering af data Der skal laves en datastruktur til at repræsentere kortet over det vejnet som undersøges. Det er et krav at dataene er let tilgængelige og dækkende. Dvs. det skal være let at udlede hvorledes forskellige kryds hænger sammen, og det skal være muligt at tegne vejnettet fuldt ud. Desuden skal den grundlæggende databaseregulering, om at samme information ikke må optræde to gange, overholdes.

Udforskning af vejnettet Den grundlæggende problemstilling er her: Hvor skal vi hen? Der skal sørges for at SMR'en får udforsket vejnettet fuldt ud.

Håndtering af odometrifejl Positionsbestemmelsen foretages udelukkende ud fra odometrimålinger. Der er uvægerligt forbundet en vis fejl med disse målinger. Dette skal der naturligvis tages højde for.

Grafisk repræsentation Et kort ville ikke være meget værd, hvis ikke det kunne vises på en forståelig måde.

Kørsel med flere SMR'er Hvad er gjort for at koden kan håndtere flere SMR'er. Hvilke antagelser er taget for at afgrænse denne problemstilling?

For at begrænse opgavens omfang, afgrænses vejnettet til at kun at indholde kryds hvis sideveje går i retning af Nord, Syd, Øst eller Vest. Desuden antages der, at der i miljøet ikke er forhindringer (eks. vægge) der skal håndteres. Dvs. funktioner til at detektere forhindringer, og f.eks. at køre uden om disse, er ikke implementeret. Dette gælder også mht. til andre robotter på samme bane. Se afsnit 2.7 for information omkring de begrænsninger, der er fortaget i forbindelse med kørsel med flere SMR'er.

2.1 Realtidsproblematikken

Da styring af SMR'en håndteres af SMRDemo via SMR-CL kommandoer, men den overordnet kortlægning og styring foregår i Jess, skal man være opmærksom på realtidsproblemer. SMRDemo håndterer SMR'ens bevægelser og sensor aflæsninger og er dermed 'opdageren', hvorimod Jess er 'planlæggeren'. Hvis planlæggeren vil have at opdageren skal stoppe når en forhindring ses, kan det gøres på to måder: Enten skal opdageren sige til, når forhindringen ses, hvorefter at planlæggeren giver besked om at stoppe. Eller også skal planlæggeren sige, at der skal stoppes når forhindringen ses. Første metode gør at opdageren i princippet fortsætter når forhindringen er set, hvorefter den får beskeden stop og stopper. Denne forsinkelse er ikke ønskværdig, fremfor hvis opdageren allerede viste, at der skulle stoppes ved en forhindring.

Kommunikationen mellem Jess og SMRDemo kan betegnes som relativ langsom. I det tilfælde, at der for en SMR opstår forhindringer, som kan skade robotten, er det vigtigt med hurtig håndtering af dette. Som lignende nævnt ovenfor kan et eksempel være, at en SMR skal stoppe, hvis ikke der er mere vej at følge, i form af en afgrund forude. Hvis den allerede ved at den skal stoppe ved afgrunden, er det at foretrække, i stedet for at vente på en kommando når den har set afgrunden.

I dette projekt håndteres realtidsproblemer ved, at betingelserne for at stoppe, når en forhindring ses, sendes i SMR-CL kommandoen og derved udføres i SMRDemo. Hvis Jess skal håndtere dette, kan der opstå så stor en forsinkelse at SMR'en ikke stopper hurtigt nok.

2.2 Grundlæggende kørsel

Når en SMR kører ad den sorte linje og ser en sidevej(e), skal den registrere sidevejen(e), samt om der evt. også er en vej frem. Det er vigtigt, at sidevejene angives korrekt i forhold til vejkrydset mht. deres retning. Det kryds, som SMR'en var ved sidst, skal naturligvis kædes sammen med det nyopdagede vejkryds via den sidevej, der peger i retning mod det forrige vejkryds. En vej kan dermed defineres, og SMR'en kan køre videre. Hvis et vejkryds besøges flere gange, skal dets data opdateres, således krydset får en mere præcis position og får tilknyttet flere omkring liggende vejkryds til sidevejene. Opdagelsen af et nyt vejkryds kan beskrives ved følgende pseudokode:

1. Et vejkryds er blevet detekteret. Gem position.
2. Tilføj sideveje til vejkryds.
3. Kør frem og se om der er en vej ligefrem.
4. Hvis der er, så tilføj den som en sidste sidevej.

Det er her vigtigt, at retningen på en sidevej er den samme uanset hvilken retning en SMR ser den fra. Hvis en SMR'en ser en sidevej og senere ser samme sidevej, men fra den modsatte retning, kan retningen være både 90° og -90° i forhold til SMR'en selv. Der vælges derfor at lave en omregning af sidevejenes vinkler til absolutte værdier i forhold til kortet.

2.3 Strukturering af data

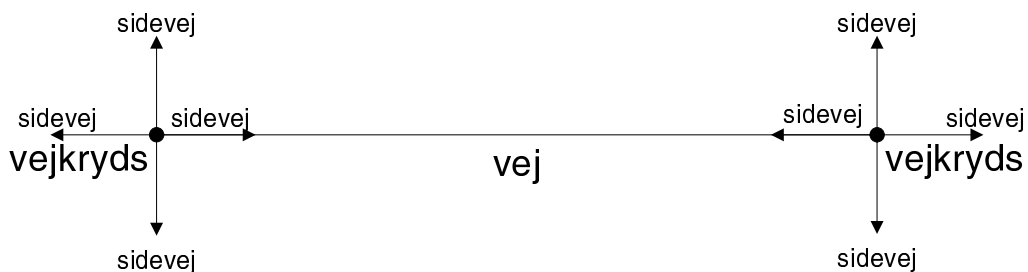
En udfordring er, at lave en hensigtsmæssig struktur af de data der skal bruges til at beskrive kortet. Dataene skal struktureres således, at kortets integritet altid bevares. F.eks. hvis koordinaterne for et vejkryds ændres, skal kortet stadig hænge sammen. Derfor skal den grundlæggende databaseregulering, om at samme information ikke må optræde mere end en gang, overholdes. F.eks. må afstanden mellem to vejkryds ikke gemmes, da denne er implicit givet ved de to vejkrydses koordinater. Ændres vejkrydssets koordinater, uden også at ændre værdien, der indeholder afstanden mellem dem, er der ikke længere integritet i dataene, og man kan ikke vide hvilke data (koordinater eller afstand) der er korrekte.

Hvis et vejkryds allerede er detekteret, og det detekteres igen, skal der naturligvis ikke registreres to vejkryds i databasen, men det eksisterende kryds skal blot opdateres. Efter at et vejkryds og dets sideveje er detekteret, som beskrevet i foregående afsnit 2.2, eksekveres følgende pseudokode:

Et eksisterende vejkryds benævnes med indekset 1, og et netop detekteret vejkryds benævnes med indekset 2.

1. Hvis kryds₂ ligger tæt på kryds₁, således at deres fejlradi overlapper hinanden, så...
2. Opdatér data for kryds₁ med data fra kryds₂. Se afsnit 2.5.
3. Slet kryds₂ og dets sideveje.
4. Ellers...
5. Registrer da kryds₂ som et gyldigt kryds.
6. Forbind den sidevej SMR'en kom til med den sidevej SMR'en forlod fra sidste vejkryds.

Denne funktion er meget vigtig, da det er hensigten at konstant opdatere de data der beskriver kortet. Følgende figur 2.1 viser to vejkryds.



Figur 2.1: To vejkryds med sideveje

Strukturen bliver da, at et kryds har fire tilhørende sideveje. Hver sidevej er, via en vej, sammenkædet med en sidevej tilhørende et andet kryds. Omvendt kan det formuleres, at en vej har to ender (sideveje), som hver er tilknyttet et kryds.

Koden bør desuden struktureres således, at den kan håndtere, at mere end én SMR-robot udforsker samme vejnet.

2.4 Udforskning af vejnettet

For at sikre, at hele vejnettet bliver udforsket, skal der implementeres en funktion, som bestemmer hvor SMR'en skal køre hen. Der er grundlæggende to scenarier hvori robotten skal bestemme sin rute:

- Der er mindst et kryds tilbage, der indeholder en udforsket sidevej.
- Der er ikke flere udforskede sideveje tilbage.

Der er mindst et kryds tilbage, der indeholder en udforsket sidevej

Her skal SMR'en finde det nærmeste kryds, med en udforskede sidevej, og køre ud af denne sidevej. Her er der to muligheder: Enten er der en udforsket sidevej i det kryds hvori SMR'en står i. Ellers er der ikke.

I det første tilfælde er det let at se at SMR'en ikke skal køre noget sted hen, men blot vælge en af de udforskede sideveje krydset indeholder. Hvilken sidevej er ikke yderligere specificeret.

I det tilfælde af at det kryds SMR'en står i ikke har udforskede sideveje, skal nærmeste kryds med udforskede sideveje findes. Dette gøres vha. af (en let modificeret) *Dijkstras algoritme*. Dijkstras algoritme er en grundlæggende og velkendt algoritme til at kortlægge korteste afstande fra et udgangspunkt til andre punkter i en graf. Til hvert punkt associerer algoritmen en værdi, der repræsenterer afstanden fra udgangspunktet til det pågældende punkt. Punkterne er i dette projekt vejkyds.

Det er i dette projekt ikke funktionelt at associere afstandene til vejkydsene selv, idét det skal være muligt for flere SMR'er kan udregne deres ønskede ruter uden at genere hinanden. Derfor modificeres algoritmen til at bruge lister indeholdende undersøgte vejkyds. Rækkefølgen af disse lister bliver så igen vedligeholdt vha. af en liste.

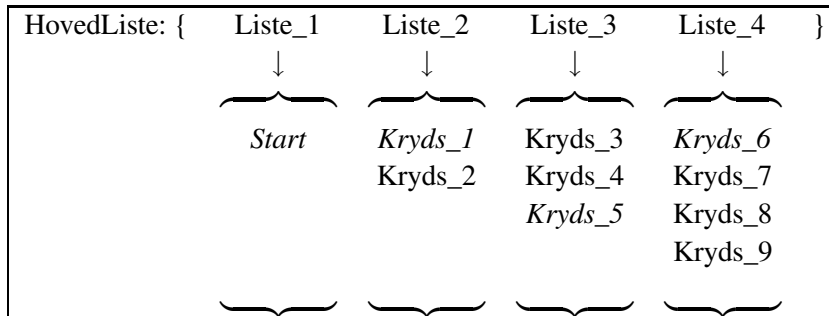
Algoritmen forudsætter at det punkt SMR'en står i er undersøgt (og ikke indeholder udforskede sideveje) og er indsat i Liste_1. Liste_1 er indsat i HovedListen. Algoritmen vil da se således ud:

1. Sæt L = den sidste liste i HovedListen.
2. Find vejkyds der er forbundet med vejkyds i L.
3. Tilføj fundne fejkyds til ny Liste_# og indsæt denne i HovedListen.
4. Hvis et af de fundne vejkyds har en udforsket sidevej...
5. ⇒ Vælg da dette kryds som destination og afslut algoritmen.

6. Ellers...

7. ⇒ Gå til 2.

Listestrukturen vil efter udførelse af algoritmen være (Start er det punkt hvori SMR'en står):



Hvis det antages at det nærmeste kryds med udforskede sideveje er *Kryds_6*, skal der nu findes en rute fra *Start* til *Kryds_6*. Dette gøres vha. backtracking, hvor der rekursivt søges efter veje, der forbinder det kryds, algoritmen sidst undersøgte, med et kryds fra forrige liste. Dette illustreres bedst med et eksempel:

1. Der skal først findes en vej der går fra *Kryds_6* til et kryds i *Liste_3*. Denne vej fører til *Kryds_5*.
2. Der skal dernæst findes en vej der går fra *Kryds_5* til et kryds i *Liste_2*. Denne vej fører til *Kryds_1*.
3. Til sidst skal der findes en vej der går fra *Kryds_1* til *Start*. Det er ud af denne vej, SMR'en skal køre.

Således er korteste vej til nærmeste kryds med udforskede sideveje bestemt. Det bemærkes dog, at korteste vej er i betydningen af antal kryds SMR'en skal passere – ikke den egentlige distance.

Alle listerne i algoritmen er tilknyttet en enkelt SMR. Det betyder at hver SMR har sit eget datasæt til brug ved udførelsen af algoritmen og de generer derfor ikke hinanden ved samtidig kørsel af algoritmen.

Der er ikke flere udforskede sideveje tilbage

I tilfælde af at der ikke er flere udforskede sideveje tilbage, skal SMR'en køre hen til det kryds der har været besøgt færrest gange. Ved at genbesøge kendte kryds vil fejlen tilknyttet et kryds kunne reduceres (se afsnit 2.5). Da det ikke er utvetydigt hvilket kryds der er besøgt færrest gange (flere kryds kan godt være besøgt samme antal gange), er det igen det kryds der er tættest på, der har størst interesse. Selve algoritmen er i blot 2 trin:

1. Find det mindst antal gange (n_{\min}) et givet kryds har været besøgt.
2. Kør algoritmen fra forrige afsnit, hvor stopbetingelsen for algoritmen nu er, at det fundne kryds er besøgt n_{\min} gange.

I denne algoritme ses der bort fra det kryds hvori SMR'en står, da det ikke er muligt at genbesøge det kryds man selv står i.

2.5 Håndtering af odometrifejl

Da positionsbestemmelse udelukkende foretages ud fra odometrimålinger, og der i forbindelse med disse målinger altid vil være tilknyttet en vis fejl, vil et kryds sandsynligvis aldrig blive opmålt til at være 100% samme sted, når det bliver genbesøgt. Det er derfor nødvendigt at definere en fejl for vejkrydsene, således at et vejkryds ikke kun udgør et koordinat men et koordinat og en usikkerhed.

Usikkerheden for et kryds kan tolkes som en radius omkring et koordinatsæt, hvori krydset kan befinde sig. Denne radius bruges i forbindelse med håndtering af fundne vejkryds. Dvs. til at bestemme hvorvidt en nyt kryds er fundet, eller om det er et kendt kryds, der er genbesøgt. Se afsnit 2.3 for yderligere analyse.

Der er grundlæggende to tidspunkter, hvor der skal beregnes usikkerhed: Under kørsel fra et kryds til et andet og når to sammenfaldende kryds (et nyt og et kendt) slås sammen til ét kryds.

Under kørsel fra et kryds til et andet vil der blive akkumuleret en usikkerhed i odometrien. Denne usikkerhed beregnes som en funktion af SMR'ens tilbagelagte distance fra sidst besøgte kryds, samt dets usikkerhed. Dvs. desto længere SMR'en kører, desto større antages usikkerheden for odometrien at være. Tænkes det at SMR'en er kørt fra et kryds med indeks 1 til et kryds med indeks 2, da vil fejlen blive beregnet ud fra følgende formel:

$$u_2 = \sqrt{\text{errFactor} \cdot \text{dist} + u_1^2} \quad (2.1)$$

hvor 'u' står for usikkerhed, 'dist' er distancen imellem punkterne og 'errFactor' er en proportionalitetskonstant.

Dette skaber en progression af usikkerheden, der er hurtig, når usikkerheden er lille, og langsommere når den er stor. Der er desuden defineret en øvre grænse for usikkerheden, idét den aldrig må blive så stor, at usikkerheden for to vejkryds overlapper hinanden.

Når et nyt og et kendt vejkryds skal slås sammen, skal der ske en vægtning af det nye kryds' koordinater i forhold til det kendte kryds' koordinater. Vægtningen skal foretages på baggrund af krydsenes placering, deres usikkerhed og hvor mange gange det kendte kryds har været besøgt. Tænkes det, at et nyt kryds (kryds_n) skal slås sammen med et kendt kryds (kryds_k), bliver det kendte kryds' koordinater opdateret som følger:¹

$$u_d = \frac{\text{distance}_{1 \rightarrow 2}}{2}$$

$$x_k = \frac{n_k \cdot u_d \cdot x_k + x_n \cdot u_k}{n_k \cdot u_d + u_k} \quad (2.2)$$

hvor 'n' er det antal gange krydset har været besøgt, 'u' er usikkerheden for krydset og 'u_d' en usikkerhed beregnet på baggrund af distancen imellem de to kryds.

¹Her blot vist for en koordinat – eks. x-koordinaten. Det samme gør sig gældende ved y-koordinaten.

Herved opnås en vægtning af de to kryds' koordinater på baggrund af det kendte kryds' usikkerhed, samt afstanden imellem de to kryds. Når det er u_d , der er brugt, i stedet for usikkerheden for det nye kryds, , skyldes det at når SMR'en returnerer til et kryds og rammer krydsets placering meget præcist, kan man formode at placeringen af krydset er forholdsvis korrekt. Det er netop halvdelen af distancen imellem de to punkter der udgør den maksimale (gennemsnitlige) usikkerhedsradius for de to punkter. Det er således muligt at tilskrive det kendte kryds en mindre usikkerhed end tidligere.

Udregningen af ny usikkerhed sker så blot ved en vægtning af usikkerhederne:

$$u_k = \frac{u_k \cdot n_k + u_d}{n_k + 1} \quad (2.3)$$

2.6 Grafisk repræsentation

For at et kort er noget værd, skal det kunne repræsenteres grafisk, således at mennesker let kan tyde det. Derfor opstilles følgende krav til et brugerinterface:

- Det skal til et hvert tidspunkt vise det komplette indhold af kort-databasen.
- Kryds skal markeres med deres usikkerhed.
- Sideveje (udforskede samt udforskede) skal vises.
- Veje imellem krydsene skal vises.

Derudover er der et ønske om at et kryds' adresse i databasen, samt det antal gange det pågældende kryds har været besøgt, skal vises. Dette er mest benyttet til at debugge systemet. Derudover er det ønskeligt at få skrevet en status for hver SMR, i form af hvor den ønsker at køre hen.

2.7 Kørsel med flere SMR'er

For at kunne håndtere flere SMR'er skal nogle af dataene være robotspecifikke. Kortet må ikke være det, da dette på alle tidspunkter skal kunne tilgås af én eller flere SMR'er. Derimod skal data såsom 'hvor en specifik SMR har været', 'hvor stor dens odometrifejl er' og 'hvor langt den har kørt' være tilknyttet en specifik SMR. Når regler derefter evalueres, er det muligt at betinge at reglen eksekveres for den rigtige SMR.

Eksempelvis hvis en SMR er ved at udforske et bestemt vejkryds, vil en regel håndtere krydsets position ud fra SMR'ens odometri. Når det derefter skal forbindes med det 'sidst besøgte vejkryds', er det vigtigt at det 'sidst besøgte vejkryds' er knyttet til den bestemte SMR. Hvis flere SMR'er hver laver et fact med 'sidst besøgte vejkryds' er det svært at vide hvilket vejkryds, som skal forbindes til det der udforskes, og en vej kan derfor blive placeret helt forkert på kortet.

Et andet aspekt i forbindelse med at køre med flere SMR'er er, at de kan møde hinanden. Da der i dette projekt er set bort fra evt. forhindringer der kan forekomme på banen, er der heller ikke taget højde for

at de kan møde hinanden. For at kunne teste, at der kan køre flere SMR'er samtidigt, kan der startes flere visualizere med identiske kort og med hver en SMR. Jess kan så styre alle SMR'erne og få input fra dem alle, men de vil aldrig kunne møde hinanden. De vil dog formentlig (i hvert fald efter et stykke tid) komme til at følge hinanden rundt på banen, idét det er det samme kort og samme algoritmer de kører efter.

Implementering 3

Her vil implementeringen blive gennemgået. Gennemgangen vil i store træk følge analysen.

3.1 Realtidsproblematikken

Som beskrevet i analysen, vil SMRDemo blive brugt til at håndtere tidskritiske funktioner mens Jess vil blive brugt til at håndtere planlægningsmæssige funktioner. Den grundlæggende opgave for SMR'en, er at den skal følge en sort linie ved at benytte lyssensorerne. Linien skal følges indtil den kommer til et kryds, eller den er nået til enden af en blind vej. Et kryds detekteres ved at de to yderste lyssensorer til højre og/eller venstre "se" noget mørkt. Enden af en blind vej detekteres ved at ingen af lyssensorerne ser noget mørkt. Det, at en lyssensor ser noget mørkt, er defineret ved, at den lysstyrke lyssensoren måler er mindre end værdien SmrTh (SMR Line Threshold) som ligger i template SmrVar.

SMR-CL Kommandoen til af følge en sort linie kommer derved til at se således ud:

```
followline bm : ((( $line0 < SMRth & $line1 < SMRth ) | ( $line6 < SMRth & $line7 < SMRth ))
  | ( ( $line0 > SMRth & $line1 > SMRth & $line2 > SMRth & $line3 > SMRth & $line4 >
    SMRth & $line5 > SMRth & $line6 > SMRth & $line7 > SMRth ) )
```

Dette kode er implementeret vha. to hjælpefunktioner; *SMRcheckForIntersection* og *SMRcheckForNoLine*:

```
(deffunction SMRcheckForIntersection()
  (format nil "(( $line0 < SMRth & $line1 < SMRth ) | ( $line6 < SMRth & $line7 < SMRth
  ))")
)

(deffunction SMRcheckForNoLine()
  (format nil "( $line0 > SMRth & $line1 > SMRth & $line2 > SMRth & $line3 > SMRth &
  $line4 > SMRth & $line5 > SMRth & $line6 > SMRth & $line7 > SMRth )" )
)
```

Derved bliver koden der afsender kommandoen til SMRDemo mere overskuelig:

```
(SMRTalk2 (format nil "followline bm : (%s) | (%s)" (SMRcheckForIntersection) (
  SMRcheckForNoLine)) ?SmrVar)
```

SMRTalk2 håndterer et andet aspekt af realtidsproblematikken, da den sørger for at slottet AllowTalk i SmrVar bliver sat til 'no'. Regler der skal kommunikere med SMRDemo, er dernæst betinget af at AllowTalk er sat til 'yes':

```
?SmrVar<-(SmrVar (AllowTalk yes) ...)
```

AllowTalk bliver sat til 'yes' af reglen *SMRTalkAllowed*, der matcher på at kommandokøen er tom (id'et for *commandqueue* og *currentcommand* er ens) og at kommandoen er udført (status er lig "done 0"):

```
(defrule SMRTalkAllowed
  ?SmrVar<-(SmrVar (robot ?r) (AllowTalk no))
  (currentcommand (status "done 0") (id ?id) (robot ?r))
  (commandqueue (id ?id) (robot ?r))
=>
  (modify ?SmrVar (AllowTalk yes) (calcDir yes))
)
```

3.2 Strukturering af data

Til implementeringen af kort-databasen, er der defineret følgende 3 templates:

Vertice Et Vertice er et **vejkryds**. De vigtigste data i et Vertice består er dets koordinater og fejlfaktor:

```
1 (deftemplate Vertice
2   (slot x)
3   (slot y)
4   (slot estErr)
5   (slot status)
6   (slot timesVisited)
7 )
```

LineEnd En LineEnd er en sidevej til et kryds. LineEnds bliver knyttet til Vertices vha. af slottet Vertice:

```
1 (deftemplate LineEnd
2   (slot Angle)
3   (slot Vertice)
4 )
```

Line En Line er vejen imellem to sideveje (LineEnds). Den har kun et slot: LineEnd. Selvom det er et multislot, er det meningen at slottet LineEnd kun skal indeholde 2 LineEnd-facts:

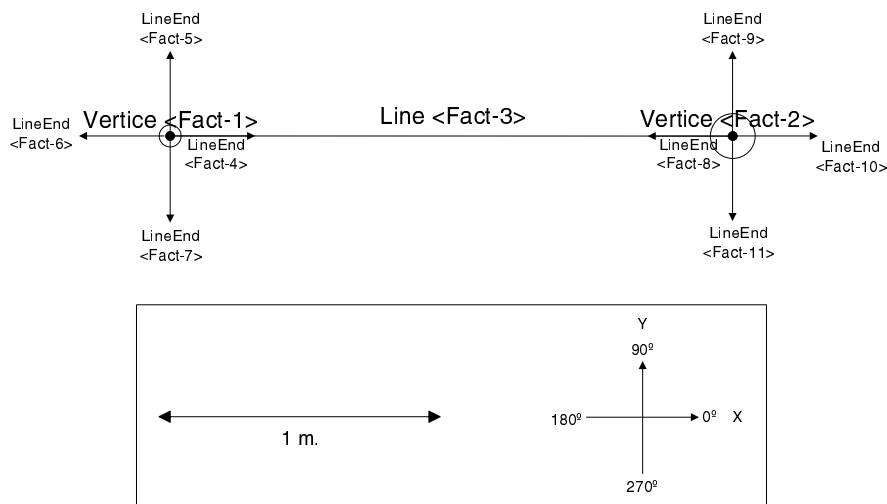
```
1 (deftemplate Line
2   (multislot LineEnd)
3 )
```

Dermed kan kortet repræsenteres i form af Vertices (kryds), med bestemte koordinater og en fejlfaktor; LineEnds som angiver retningen af sideveje og Lines der knytter LineEnds sammen. Eksemplet i figur 3.1 viser sammenhængen mellem et sæt Vertices, LineEnds og en Line.

Minikortet på figur 3.1 er beskrevet vha. følgende facts¹:

```
<Fact-1> (Vertice (x 0) (y 0) (estErr 0.01) (status FullBlood) (timesVisited 1))
<Fact-2> (Vertice (x 2) (y 0) (estErr 0.02) (status FullBlood) (timesVisited 1))
<Fact-3> (Line (LineEnd <Fact-4> <Fact-8>))
<Fact-4> (LineEnd (Angle 0) (Vertice <Fact-1>))
<Fact-5> (LineEnd (Angle 90) (Vertice <Fact-1>))
<Fact-6> (LineEnd (Angle 180) (Vertice <Fact-1>))
<Fact-7> (LineEnd (Angle 270) (Vertice <Fact-1>))
```

¹Dette er et illustrativt eksempel. Den kronologiske rækkefølge af facts'ne er ikke korrekt.



Figur 3.1: Eksempel på opbygning af en vej

```
<Fact-8> (LineEnd (Angle 180) (Vertice <Fact-2>))
<Fact-9> (LineEnd (Angle 90) (Vertice <Fact-2>))
<Fact-10> (LineEnd (Angle 270) (Vertice <Fact-2>))
<Fact-11> (LineEnd (Angle 0) (Vertice <Fact-2>))
```

Det ses at de to Vertices (fact 1 og 2) ikke har lige store fejlfaktorer. Dette er vist på figur 3.1, ved at de to Vertices er omkredset af cirkler med forskellige radii.

Vertices status

For at sikre at reglerne benytter et Vertice korrekt, har hver Vertice fået tildelt en status. Herved kan det sikres, at der f.eks. ikke navigeres efter et Vertice, der endnu ikke er blevet forenet med et eksisterende Vertice, og at Verticet ikke bliver forenet før det er klar til at blive det.

Som kuriositet er der valgt at benytte en rocker-terminologi til at beskrive status. Det resulterer i at et Vertice kan antage én af følgende fire værdier:

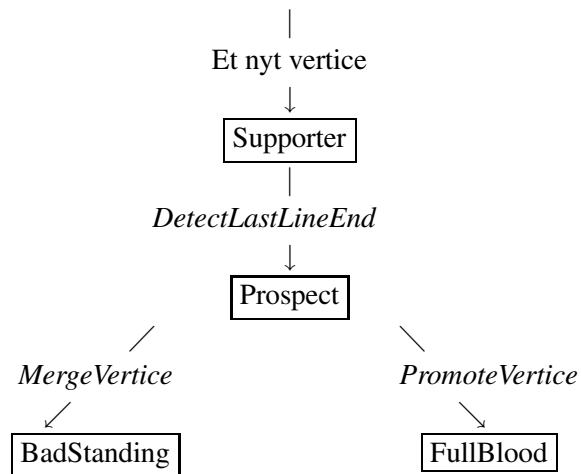
Supporter Dette er et Vertice, der er ved at blive udforsket af en SMR. Dvs. SMR'en er endnu ikke færdig med at tilføje LineEnds til Verticet. Så længe status er Supporter, bliver det ikke kontrolleret om Verticet ligger oven i et eksisterende Vertice.

Prospect Når en SMR er færdig med at udforske et Supporter-Vertice, ændrer reglen *DetectLastLineEnd* dets status til Prospect. Når et Prospect-Vertice dukker op, bliver en af to regler eksekveret: Enten ligger Prospect-Verticet tæt på et FullBlood-Vertice, hvorefter *MergeVertice* kombinerer de Vertices positionsdata sammen. Ellers eksekveres *PromoteVertice*, som hæver det nye Vertices status til FullBlood.

FullBlood FullBlood den højeste status et Vertice kan opnå. Kun FullBlood-Vertices er gyldige vejkryds i kortet.

BadStanding Et BadStanding-Vertice er et Vertice der skal slettes. Et Prospect-Vertice får dette status af *MergeVertice* hvis det ligger oven i et eksisterende FullBlood-Vertice.

Et Vertices liv kan altså beskrives med følgende diagram:



Forening af Vertices

Foreningen af Vertices sker vha. reglen *MergeVertices*. Den sørger for at koordinaterne og usikkerheden på FullBlood-Vertices bliver opdateret som beskrevet i afsnit 2.5.

Hvis SMR'en har kørt samme vej tidligere, eksisterer der allerede en Line der forbinder samme to LineEnds. Derfor vil der komme til at optræde en Line for meget i databasen, som så skal slettes. Dette sørger reglen *deleteDuplicateLines* for.

SMR'en ved altid hvilken LineEnd den sidst kørte ud af. Når den så detekterer et nyt Vertice, bliver den LineEnd, SMR'en er kommet til, derfor altid forbundet med en Line. Dvs. Prospect-Verticet har altid en LineEnd der er forbundet med en Line. For at denne Line ikke går tabt når Prospect-Verticet slettes, ændres Line'en således at den nu danner forbindelse til FullBlood-Verticets LineEnd med samme vinkel (angle).

Sletning af Vertices

MergeVertice kan ikke bare slette et Vertice med en retract-kommando, da der så ville være en række løse LineEnds tilbage i databasen. Derfor ændrer *MergeVertice* status til BadStanding,

Reglen *DeleteLineEnd* holder øje med, om der eksisterer LineEnds tilknyttet til et Vertice med status BadStanding. Findes sådanne LineEnds slettes disse med retract-kommandoer.

Når alle LineEnds er slettet, sørger reglen *DeleteVertice* for, at selve Verticet bliver slettet. Dette sker ved, at reglen holder øje med om der eksisterer Vertices med status BadStanding uden tilknyttet LineEnds. Gør der det bliver Verticet slettet med en med en retract-kommando.

Forfremmelse af Vertices

Hvis et Prospect Vertice areal ikke overlapper et eksisterende FullBlood Vertices areal, antages det, at dette er et nyt Vertice. *promoteVertice* sørger i dette tilfælde for at ændre Prospect Vertices status fra Prospect til FullBlood.

3.3 Grundlæggende kørsel

Den grundlæggende kørselshåndtering sker vha. reglen *Drive*, gengivet her:

```

1 (defrule Drive (declare (salience -100))
2     ?SmrVar<-(SmrVar (robot ?r) (AllowTalk yes) (State Idle) (folLineEnd ?followLE) (
3         curVertice ?curV))
4         (odometry (robot ?r) (theta ?theta) (dist ?dist))
5     =>
6         (bind ?ang (CalcDirection ?theta))
7         (bind ?wantedAng (fact-slot-value ?followLE Angle))
8         (bind ?turn (- ?wantedAng ?ang))
9
10        (SMRTalk2 "idle" ?SmrVar)
11        (SMRTalk2 (format nil "set \"$odox\" %f" (fact-slot-value ?curV x)) ?SmrVar)
12        (SMRTalk2 (format nil "set \"$odoy\" %f" (fact-slot-value ?curV y)) ?SmrVar)
13
14        (SMRTalk2 (format nil "turn %d" ?turn) ?SmrVar)
15        (SMRTalk2 (format nil "followline bm : (%s) | (%s)" (SMRcheckForIntersection) (
16            SMRcheckForNoLine)) ?SmrVar)
17        (SMRTalk2 "idle" ?SmrVar)
18        (modify ?SmrVar (LastLineEnd ?followLE) (LastDrivenDist ?dist) (State Drive))
19    )

```

Kriterierne for at *Drive* kan eksekveres er relativt få. State skal være 'idle' og AllowTalk skal være 'yes'. Det at State er lig 'idle' sikrer at SMR'en ikke er i færd med at undersøge et kryds men er klar til at køre videre. Til gengæld for disse få krav er, at salience på reglen er sat lavt, således at det er sikret at andre regler (til opdatering af kort og ønsket kørselsretning) eksekveres før SMR'en kører videre.

Linie 5 til 7 håndterer omregningen fra kortets absolutte vinkler til den relative vinkel SMR'en skal dreje til.

Linie 9 til 11 laver et lille trick, hvor SMR'ens koordinater bliver opdateret til at være de samme som koordinaterne på det kryds SMR'en forlader. Dette gøres fordi, når et kryds bliver genbesøgt, bliver dette kryds' position opdateret som en vægtning imellem nye koordinater for krydset og gamle koordinater for samme kryds. Derfor er det ikke sikkert at SMR'ens odometri angiver, at den står i det kryds den forlader.

Når followline kommandoen (afsendt af *Drive*) er udført, er SMR'en er nået hen til et kryds eller enden af en blind vej. Da sørger reglen *DetectSupportVertice*, for at et nyt Vertice med status Supporter bliver oprettet. Verticets koordinater beregnes ud fra en forskydning af SMR'ens odometri, da der skal tages højde for at Verticet befinder sig under lyssensorerne, mens SMR'ens odometri har nulpunkt midt mellems SMR'ens baghjul. Dette håndteres af funktionerne *offsetX* og *offsetY*.

DetectSupportVertice sørger også for at der bliver tilføjet en LineEnd bagud i den retning SMR'en kom fra. Denne LineEnd bliver yderligere forbundet, via en Line, med den LineEnd SMR kørte ud af da den forlod forrige Vertice. Hvis de to yderste lyssensorer til højre og/eller venstre ser noget mørkt, dvs. Verticet har en side vej til højre og/eller venstre, tilføjes også LineEnds til højre og/eller venstre.

Når alle LineEnds er detekteret afgives en kommando til SMRDemo om at køre SMR'ens egen længe frem. Dette gøres, da lyssensorerne på dette tidspunkt ikke kan se, om der er en linie lige frem, dvs. evt. et sidste LineEnd. Efter at SMR'en har kørt sine egen længde frem, undersøges det om nogen af de fire midterste lyssensorer ser noget mørkt. Er det tilfældet, tilføres en LineEnd ligefrem til Verticet.

Grunden at det netop er SMR'ens egen længde der køres frem, er at når SMR'en efterfølgende evt. skal dreje og følge en sidevej, er det er praktisk at Verticet befinder sig præcist under SMR'ens omdrejningspunkt. Derved sikres det at SMR'en altid rammer linien med de midterste lyssensorer – uanset hvilken retning den drejer til. Herefter ændres Verticets status fra Supporter til Prospect. Reglerne til behandling af Propect-Vertices tager derefter over og afgør hvorvidt Verticet skal forfremmes til FullBlood eller slettes.

3.4 Udforskning af vejnettet

En forudsætning for at kunne beslutte hvor SMR'en skal køre hen er, at den ved hvor den er. Til det formål er der lavet en regel (*posUpdate*) der løbende opdaterer slottet *curVertice* i *SmrVar*. Slottet *curVertice* i *SmrVar* repræsenterer således altid nærmeste Vertice, SMR'en tror den befinder sig ved.

Til selve udførelsen af ruteudregningsalgoritmerne, er der implementeret tre regler: *updWantedDir1*, *updWantedDir2* og *updWantedDir3*. Betingelsen for at alle tre regler kan eksekveres, er at flaget *calcDir* i *SmrVar* er sat til "yes". Dette flag bliver sat af *posUpdate* samt af reglen *SMRTalkAllowed*, der afgør om SMR'en er klar til at modtage nye kommandoer.

Reglerne sætter en værdi i slottet *folLineEnd* (Follow LineEnd) i *SmrVar*. Dette slot benyttes af reglen *Drive*, som drejer SMR'en i den ønskede retning, og giver SMRDemo kommando til at følge den sorte linie.

updWantedDir1

updWantedDir1] tager sig af det tilfælde, hvor det Vertice (kryds) som SMR'en står i, har udforskede LineEnds (sideveje). Reglen (i beskåret udgave) er angivet her:

```

1  (defrule updWantedDir1
2    ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists))
3    =>
4    (bind ?result (run-query* findLostLineEnds ?curVert))
5    (if (?result next) then
6      (modify ?SmrVar (folLineEnd (?result get LE)) (calcDir no)
7    else
8      (modify ?SmrVar (verticeLists (assert (verticeList (robot ?r) (vertices ?curVert))))))
9    )
10 )
```

Det ses, at betingelsen for at *updWantedDir1* bliver eksekveret, bl.a. er at *verticeLists* er tom. *verticeLists* er den hovedliste, der indeholder lister af *Vertices*, som på et givet tidspunkt er blevet evalueret af ruteudregningsalgoritmerne. Listen er en midlertidig variabel i *SmrVar*, der kun benyttes af ruteudregningsalgoritmerne. Når en rute er bestemt, slettes listen igen,

Queryen *findLostLineEnds* benyttes i denne regel til at undersøge, om det kryds *SMR*'en står i har udforskede *LineEnds*. Dvs. om den kan finde en *LineEnd*, som ikke er tilknyttet nogen *Line*. Hvis *findLostLineEnds* finder en *LineEnd* (linie 5), vil den pågældende *LineEnd* blive angivet som værende den som *SMR*'en skal følge (linie 6), og *calcDir* bliver sat til "no". Ellers vil *Verticet* (i form af en *verticeList* med kun et element) blive tilføjet til *verticeLists*.

updWantedDir2

updWantedDir2 håndterer det tilfælde, hvor alle sideveje i krydset, som *SMR*'en står i, er udforsket, og der skal findes en udforsket sidevej i et kryds længere væk. Betingelsen for at *updWantedDir2* eksekveres, er bl.a. at *verticeLists* ikke er tom – i modsætning til *updWantedDir1*, hvor den netop skal være tom. *updWantedDir2* er implementeret som følger (beskåret udgave):

```

1  (defrule updWantedDir2
2    ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists $? ?
      lastList))
3
4    ?fLineEnd<-(LineEnd (Vertice ?vertice))
5    (not (Line (LineEnd $? ?fLineEnd $?)))
6  =>
7    (bind ?newList (assert (verticeList (robot ?r))))
8    (modify ?SmrVar (verticeLists (fact-slot-value ?SmrVar verticeLists) ?newList))
9
10   (bind ?cont 1)
11   (bind ?vertices (run-query* findConnectedVertices ?r ?lastList))
12   (while (and (?vertices next) (= ?cont 1))
13     (bind ?ends (run-query* findLostLineEnds (?vertices get vertDst)))
14     (if (?ends next) then
15       (modify ?SmrVar (folLineEnd (calcDirToDstVertice ?SmrVar (?vertices get vertDst))) (
          calcDir no)
16       (cleanUpList ?SmrVar)
17       (bind ?cont 0)
18     else
19       (modify ?newList (vertices (fact-slot-value ?newList vertices) (?vertices get
          vertDst)))
20   )
21 )
22 )

```

Det er angivet, som betingelse, at *verticeLists* ikke er tom (linie 2). Dvs. *updWantedDir1* har været eksekveret. Derudover skal det være muligt at finde mindst én *LineEnd*, der ikke er tilknyttet en *Line*, for at undgå en forgæves søgning.

Queryen *findConnectedVertices* (linie 11) benyttes til at finde nabo-*Vertices* til de *Vertices* der er listet i den seneste *verticeList*. For hvert af disse nabo-*Vertices* (linie 12), benyttes queryen *findLostLineEnds* igen til at undersøge, om det pågældende *Vertice* har udforskede *LineEnds*. Findes en udforsket *LineEnd*, benyttes funktionen *calcDirToDstVertice*, til at bestemme hvilken retning *SMR*'en skal køre for at komme til dette *Vertice*. Desuden bliver *calcDir* sat til "no". Findes ikke udforskede *LineEnds* (Linie 19) gemmes de *Vertices*, der netop er blevet undersøgt, i en *verticeList*, som tilføjes *verticeLists*.

Det ses af ovenstående beskrivelse, at reglen *updWantedDir2* fungerer rekursivt, eftersom reglen bliver ved med at blive eksekveret indtil en udforsket LineEnd er fundet. Da reglen tilføjer de Vertices, den har undersøgt, til verticeLists, og i næste eksekvering undersøger nabo-Vertices til seneste angivet Vertices i verticeLists, opnås det, at Vertices længere og længere væk undersøges for udforsket LineEnds

updWantedDir3

Reglen *updWantedDir3* håndterer det tilfælde, at der ikke er flere udforskede sideveje. Dvs. at det er det mindst besøgte Vertice, SMR'en skal køre hen til. Reglen er implementeret som følger (i beskåret udgave):

```

1  (defrule updWantedDir3
2    ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists $? ?
3      lastList))
4
5    (not (and
6      ?fLineEnd<-(LineEnd (Vertice ?vertice))
7      (not (Line (LineEnd $? ?fLineEnd $?))))
8    ))
9    =>
10   (bind ?minVisitList (run-query* findMinvisited ?curVert))
11   (?minVisitList next)
12   (bind ?minVisitedValue (?minVisitList getInt minVisit))
13
14   (bind ?newList (assert (verticeList (robot ?r))))
15   (modify ?SmrVar (verticeLists (fact-slot-value ?SmrVar verticeLists) ?newList))
16
17   (bind ?cont 1)
18   (bind ?vertices (run-query* findConnectedVertices ?r ?lastList))
19   (while (and (?vertices next) (= ?cont 1))
20     (if (= ?minVisitedValue (?vertices getInt timesVisited)) then
21       (modify ?SmrVar (folLineEnd (calcDirToDstVertice ?SmrVar (?vertices get vertDst))) (
22         calcDir no)
23       (cleanUpList ?SmrVar)
24       (bind ?cont 0)
25     )
26     else
27     (modify ?newList (vertices (fact-slot-value ?newList vertices) (?vertices get
28       vertDst)))
29   )
30 )
31 )

```

Den ligner meget *updWantedDir2*, bortset fra, at i line 9 til 11 bestemmes blandt alle Vertice, det laveste antal gange et Vertice er blevet besøgt. Denne værdi bliver så brugt som stopbetingelse (Line 19) for den rekursive løkke. Betingelsen for at *updWantedDir3* eksekveres (line 4 til 7) er en invertering af betingelsen fra *updWantedDir2*, da det her er et krav, at der ikke findes et udforsket Vertice.

3.5 Grafisk repræsentation

Den grafiske del af projektet er håndteret i filen *gui.jss*. Den består grundlæggende af en funktion, der opdaterer det grafiske vindue, en række hjælpe-queries til at hente data ud af kort-databasen samt en regel der sørger for at grafikken bliver opdateret. Opdateringen sker hver gang et FullBlood Vertice bliver tilføjet eller ændret, eller en SMR ændrer sin position.

Selve funktionen til opdatering af grafikken (*painter*) foregår i en række trin:

1. Udregn optimal vinduestørrelse og offset for grafik.
2. Slet vinduets indhold.
3. Skriv SMR'ernes status.
4. Tegn veje (Lines).
5. Tegn sideveje (LineEnds).
6. Tegn Fejlen omkring kryds.
7. Skriv fact-adresser samt antal gange et Vertice har været besøgt.

Som eksempel på tegneoperationerne, følger her koden for at tegne fejlcirclerne:

```

1  (?graph setColor (Color.red))
2  (bind ?vertices (run-query* findVertices))
3  (while (?vertices next)
4    (bind ?err (round (* 2 ?res (?vertices getFloat estErr))))
5    (bind ?x (round (+ (* -0.5 ?err) (* ?res (+ ?offsetx (?vertices getFloat x))))))
6    (bind ?y (round (+ (* -0.5 ?err) (* ?res (+ ?offsety (* (?vertices getFloat y) -1))))))
7    )
8    (?graph drawOval ?x ?y ?err ?err)
9  )
10 )

```

Den benyttede query til at finde Vertices er yderst simpel. Her gengivet for eksemplets skyld:

```

1  (defquery findVertices
2    ?id<-(Vertice (x ?x) (y ?y) (estErr ?estErr) (timesVisited ?timesVisited))
3  )

```

Resten af delfunktionene i denne funktion er (mere eller mindre) lige så enkle. Der er dog den ene ting at bemærke, at mens SMR'erne har nulpunkt i nederste venstre hjørne (almindeligt cartesisk koordinat-system), har de grafiske funktioner nulpunkt i øverste venstre hjørne. Dvs. at alle y-koordinaterne skal ganges med -1 for at få et retvisende billede af kortet.

En anden ting der bør bemærkes om den grafiske funktion, er at den i virkeligheden er Java-kode i Jess-forklædning. Der kunne måske give bedre ydelse, hvis koden blev implementeret i Java direkte. Ligeledes kunne koden optimeres ved kun at gentegne de områder af vinduet der havde brug for det², men dette er ikke blevet prioritet i dette projekt.

3.6 Kørsel med flere SMR'er

Det er som beskrevet et mål at koden skal kunne håndtere flere SMR på samme vejnet. Det er derfor vigtigt at strukturere koden således, at reglerne ikke blander data fra forskellige SMR sammen. Hver

²I modsætning til at gentegne hele vinduet hver gang.

SMR har et unikt ID, som er indeholdt i alle de templates der indeholder data om SMR'en. Dette ID er gemt i slottet 'robot'. Værdien af ID'et svarer til navnet på den computer som SMRDemo afvikles på. Alle regler der håndterer en eller anden form for styrefunktion til SMR'en matcher derfor på dette slot, således at alle funktioner i programmet kan eksekveres for flere SMR'er samtidigt. Matchet foregår ved:

```
?SmrVar<-(SmrVar (robot ?r) ...)
```

Variablen *?r* bliver så givet med videre i eksekveringen af venstresiden af reglen.

Resultater

4

For at illustrere kodens funktionalitet, vil der her blive givet eksempler på testkørsler med softwaren. Alle testkørsler foregår vha. Visualizer.

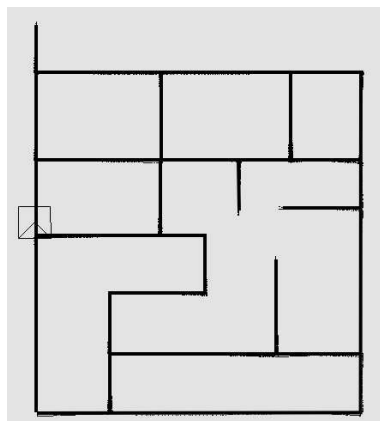
4.1 Kørsel med én SMR

SMR'en sættes til at køre på banen vist i figur 4.1. Figur 4.2 til 4.4 viser screenshots af det genererede kort, taget med passende mellemrum. Disse giver et godt indtryk af hvorledes SMR'en afdækker kortet. Tallene der skrives nær et Vertice er fact-adressen fra Jess og tallet i parentes er antallet af gange et Vertice er blevet udforsket.

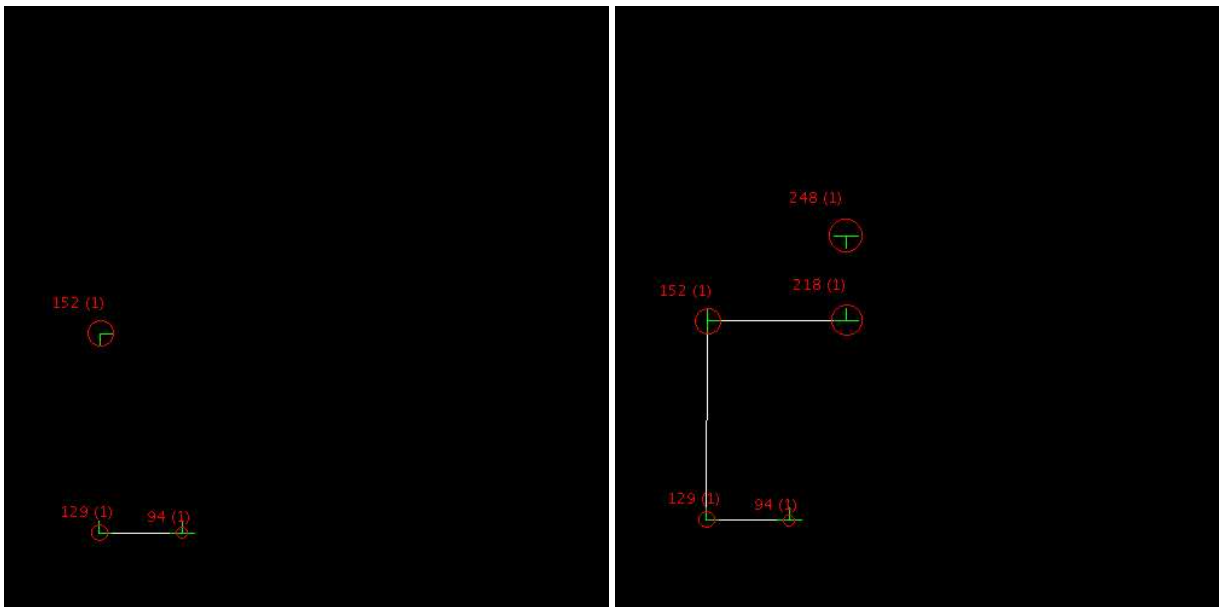
Det ses at SMR'en korrekt opdaterer kortet efterhånden som den kommer frem. For Vertices som har sideveje der ikke er udforsket, ses det, at Verticet markeres med en lille 'sidevej' som en indikator for, at der er en sidevej, der senere skal udforskes.

Cirklen omkring hver Vertice angiver usikkerheden. Det kan ses, at enkelte Vertices bliver mindre i løbet af sekvensen. Dvs. at de har været besøgt mere end en gang. Dette fremgår også af teksten i parentes nær Verticene på figuren. Dermed reduceres usikkerheden efterhånden som en SMR bevæger sig rundt og ser samme vejkryds flere gange.

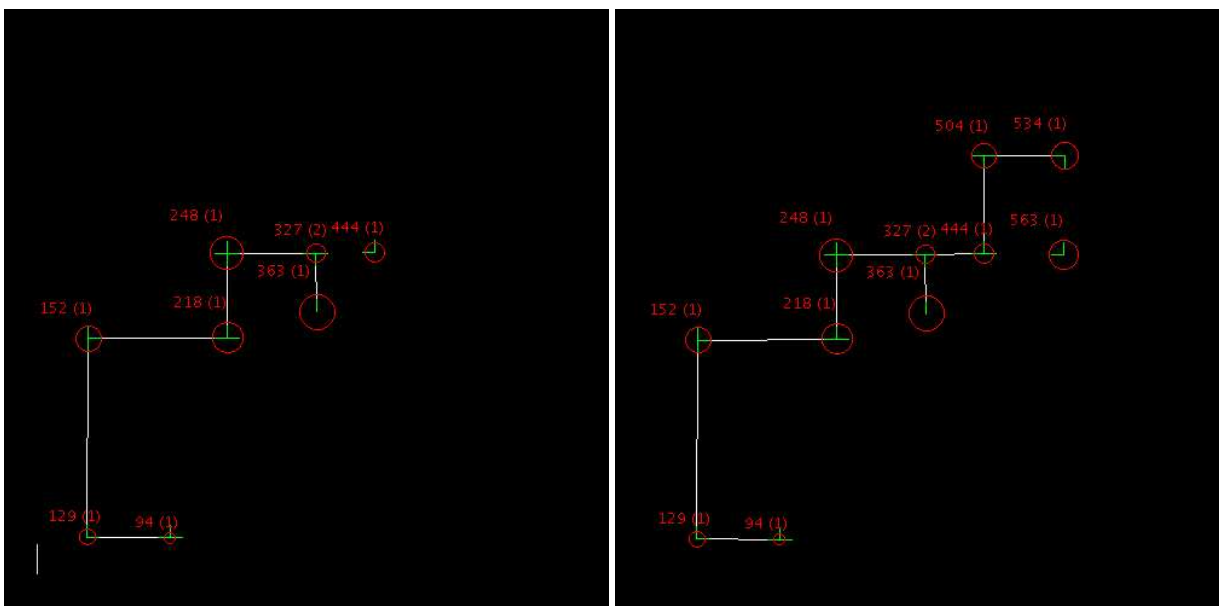
Det ses også at vejnettet bliver hensigtsmæssigt udforsket, og SMR'en hele tiden målrettet vælger at udforske nye ukendte sideveje, således at hele vejnettet hurtigst muligt bliver udforsket.



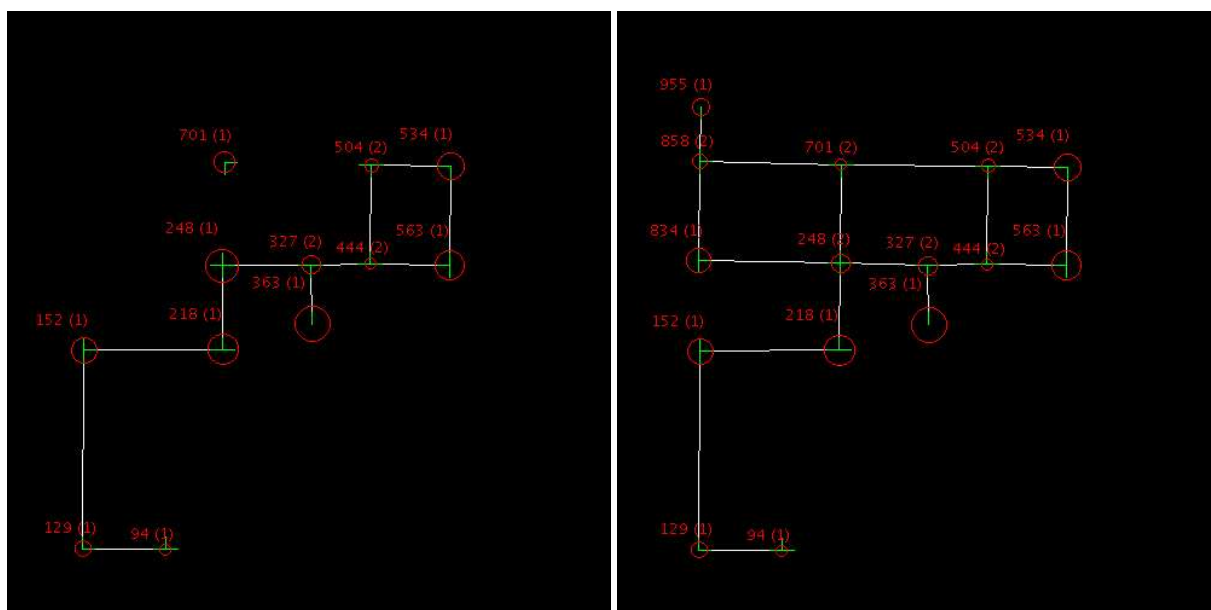
Figur 4.1: Kort til testkørsel nr. 1. Resultatet ses i figur 4.2 til figur 4.4



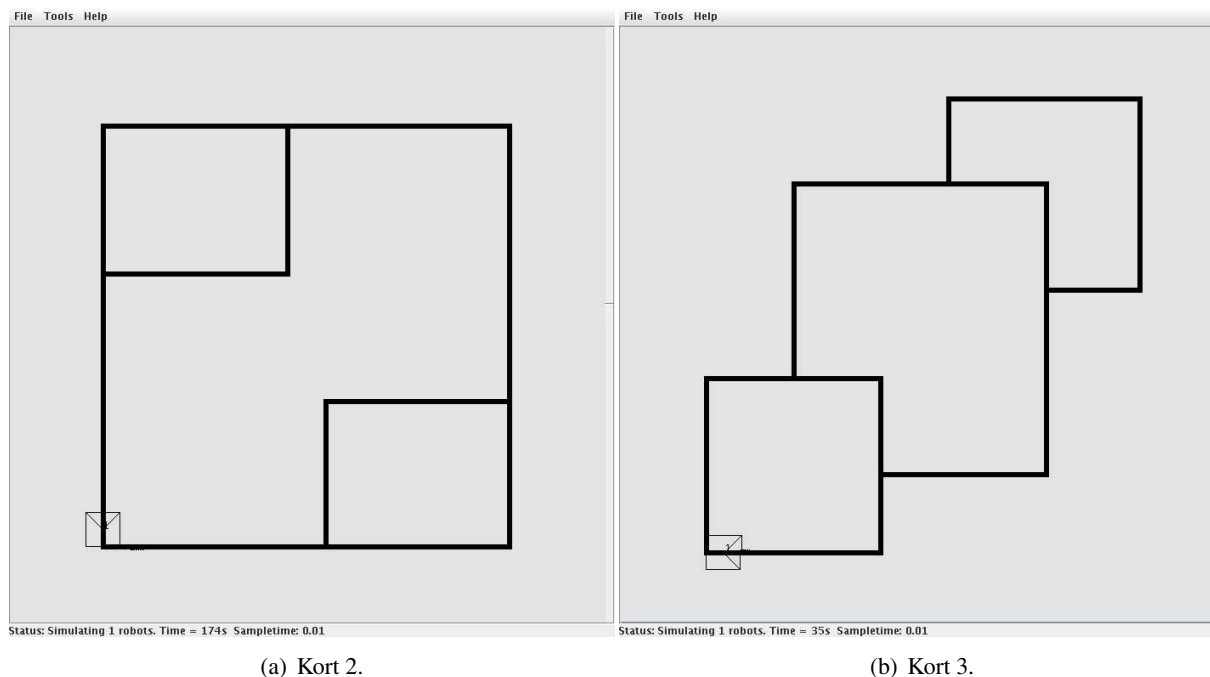
Figur 4.2: Udforskning af kortet i figure 4.1. Billede 1 og 2 ud af 6.



Figur 4.3: Udforskning af kortet i figure 4.1. Billede 3 og 4 ud af 6.



Figur 4.4: Udforskning af kortet i figure 4.1. Billede 5 og 6 ud af 6.



(a) Kort 2.

(b) Kort 3.

Figur 4.5: To andre kort som bliver udforsket i figur 4.6 og 4.7.

Figur 4.5 viser to andre kort, der er blevet udforsket af en SMR. Resultatet ses i figur 4.6 og 4.7.

Det bemærkes, at da alle kryds er fuldt udforskede, kører SMR'en nu mod det kryds der er besøgt færrest gange. Dette ses af den blå statutstekst i toppen af kortene.

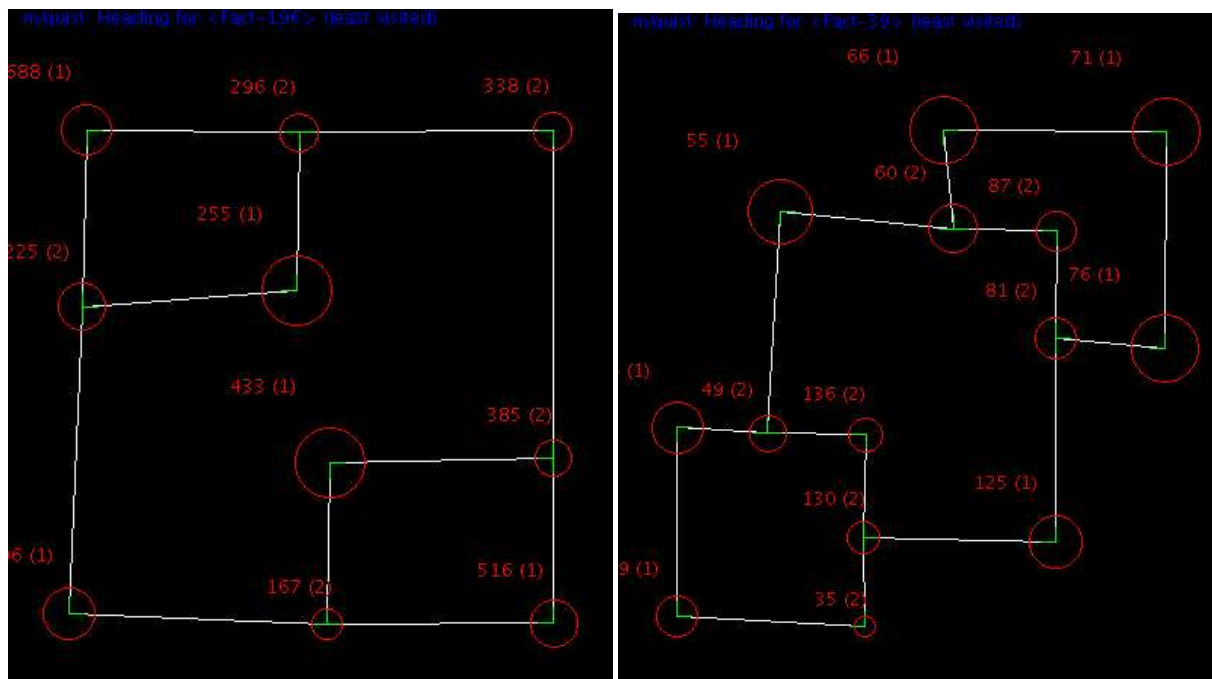
Sammenlignes figur 4.6 med 4.7, ses det at kortet i figur 4.6 har store fejl-cirkler i forhold til figur 4.7. Ydermere ser kortene i figur 4.7 også pænere ud i den forstand, at vejene er mere vinkelrette på hinanden. Fik SMR'en lov til at køre længere tid, kunne kortenes nøjagtighed forbedres endnu mere. Den ville dog aldrig blive reduceret helt ned til 0.

4.2 Kørsel med flere SMR'er

Da koden er struktureret til at kunne håndtere et vilkårligt antal SMR'er, er det testet at lade to SMR'er udforske samme vejnet samtidigt. Som nævnt er der ikke implementeret en håndtering af kollision mellem to SMR'er. Hvis en SMR møder en forhindring (om det er en anden SMR eller noget andet), har SMRDemo en indbygget mekanisme til at stoppe. Men der er ikke implementeret nogen videre håndtering af dette i Jess-koden.

For alligevel at teste funktionaliteten, startes der to Visualizere med samme kort. Der startes også to SMRDemo'er (en til hver Visualizer), som så begge styres af samme Jess-kode. SMR'erne placeres to forskellige steder på kortet som vist i figur 4.8. Dette setup svarer fuldt ud til hvis begge SMR kørte på samme vejnet (i samme Visualizer), bortset fra de kan ikke se hinanden og dermed ikke kolliderer.

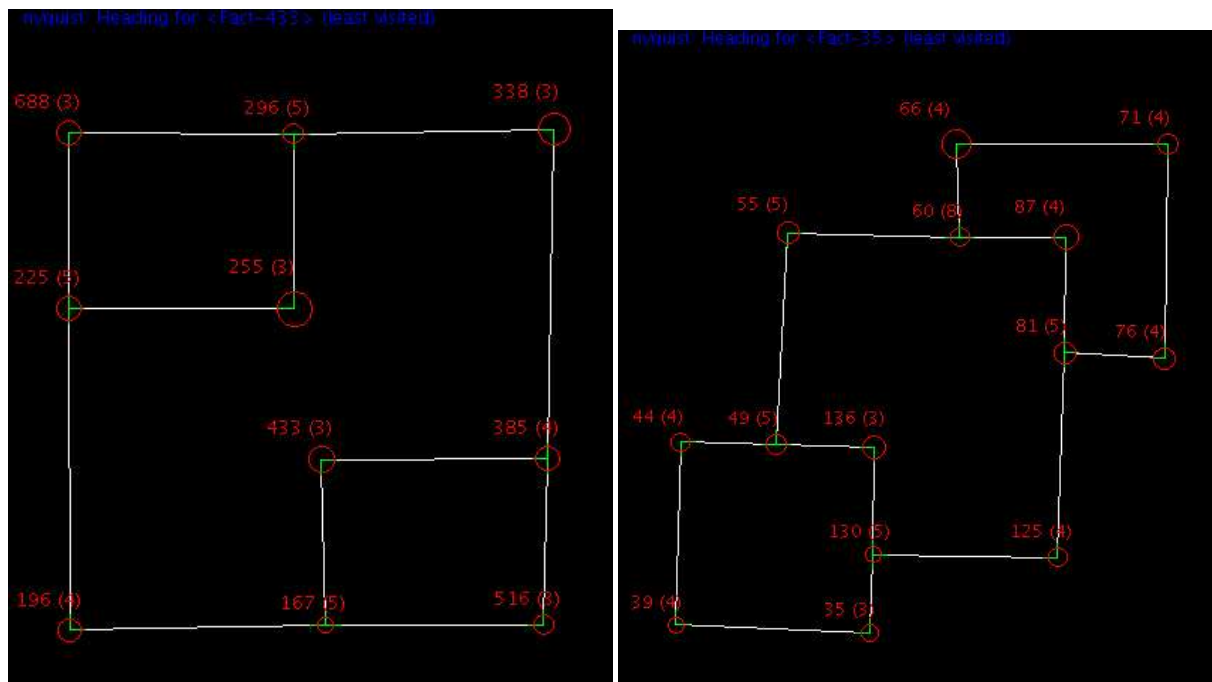
Figur 4.9 og 4.10 viser resultatet af testkørslen med to SMR'er. Det ses at SMR'erne individuelt kan opdatere samme kort. Brugen af to SMR'er øger hastigheden hvormed kortet udforskes med en faktor 2



(a) Kort 2.

(b) Kort 3.

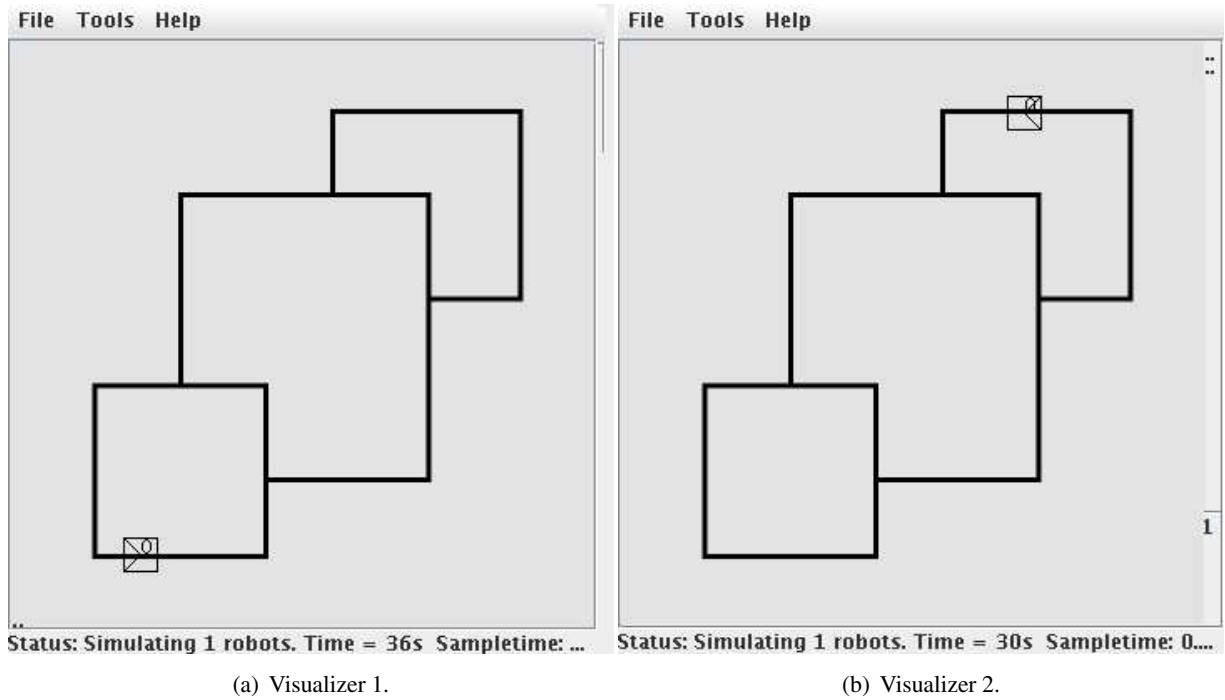
Figur 4.6: Kortene i figur 4.5 hvor hvert kryds er udforsket mindst én gang.



(a) Kort 2.

(b) Kort 3.

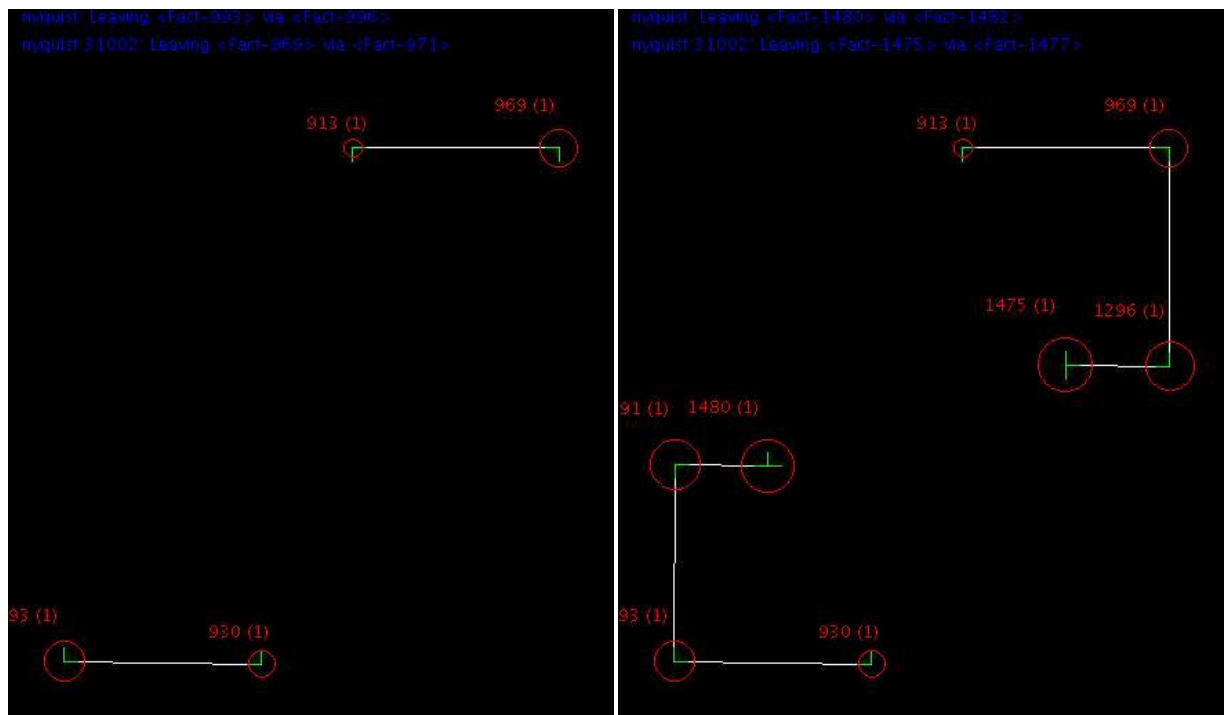
Figur 4.7: Kortene i figur 4.5 hvor hvert kryds er udforsket flere gange.



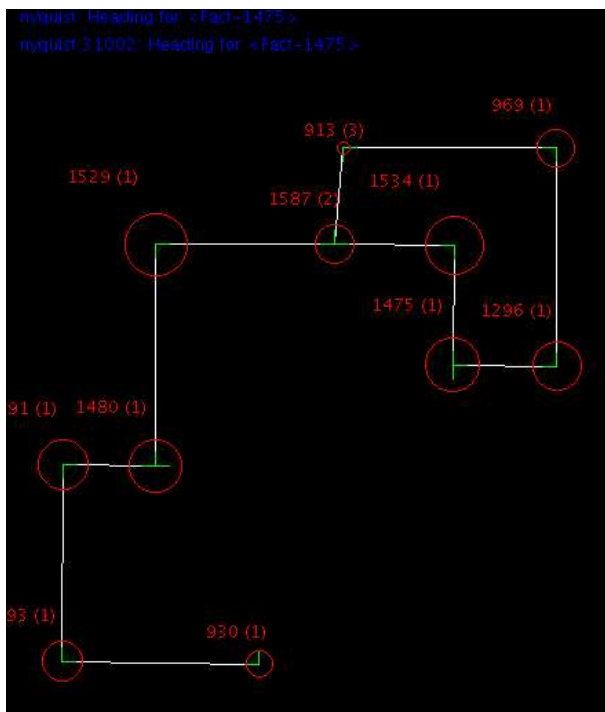
(a) Visualizer 1.

(b) Visualizer 2.

Figur 4.8: Placering af SMR'er i hver sin Visualizer med samme kort.



Figur 4.9: Resultatet af kørsel med 2 SMR'er som illustreret i figur 4.8.



Figur 4.10: Resultatet af kørsel med to SMR'er som illustreret i figur 4.8.

(der er 2 SMR'er). Denne fordel forsvinder dog, når de to SMR'er møder hinanden (dog stadig uden at kunne se hinanden). Som tidligere omtalt vil SMR'erne nu konstant vælge den samme rute at udforske kortet. Dette ses i figur 4.10, hvor statusteksten angiver, at de begge er på vej til krydset med fact-adressen 1475.

Forbedringer



Skulle der bygges videre på dette projekt, bør det overvejes at lave forskellige ændringer og forbedringer:

Håndtering af forhindringer Hvis en SMR møder en forhindring eller en anden SMR, skal den vende om og (via. kort-dataene) finde en anden destination.

Bedre samarbejde mellem flere SMR'er Da alle SMR'er på vejnettet benytter samme regler og algoritmer, vil de følge samme rute rundt på vejnettet. Dette er, som vist, uhensigtsmæssigt. Derfor skal hver SMR sætte et flag, som angiver at SMR'en har bestemt at udforske et givet Vertice. Vha. af dette flag kan andre SMR'er så undgå at køre hen til samme destination.

Flere mulige vinkler for sideveje For at afgrænse opgavens omfang, er det bestemt at vejkryds kun kan have vinkelrette sideveje. Således er det let at matche sidevejes vinkler. For at håndtere ikke vinkelrette veje, kunne en matching af en vinkel implementeres med et interval. Lidt i samme stil som det er gjort med koordinaterne i dette projekt.

Bedre funktioner til forening af Vertices Indtil videre er funktioner til at bestemme usikkerheder og Vertice's koordinater bestemt af mere eller mindre intuitive iagttagelser. Dette kunne godt rodfæstes bedre med en mere korrekt matematisk/statistisk fremgangsmåde.

Håndtering af ikke rette veje For at håndtere veje som ikke er rette veje, kan en vej evt. opdeles i flere små vejstykker. Opdelingen i x antal stykker kan ske som en funktion af SMR'en kørselsretning når SMR'en følger vejen og drejer. Derved kunne SMR'en følge og gengive krumme veje som stykvis rette linier.

Håndtering af et dynamisk vejnet Koden kan pt. ikke håndtere at vejnettet ændrer sig under kørsel. Dette skyldes, at et Prospects sideveje bliver slettet, når det forenes med et FullBlood Vertice. I stedet burde også sidevejene tages i betragtning. Dermed kunne der opnås en "second opinion" om vejkrydsenes udseende.

Konklusion

Der er redegjort for hvorledes et program, opbygget i Jess, får en eller flere SMR til at kortlægge et vejnet. Programmet er via SMR'erne i stand til at følge en sort linie og detektere vejkryds. Krydssets sideveje detekteres, samt hvordan disse er forbundet til andre vejkryds. Dataene er struktureret på en måde i databasen, så vejenes sammenhæng let kan udledes.

Til illustration for brugeren, er der opbygget et grafisk vindue, der viser kortet, som ligger i databasen. Grafikvinduet viser SMR'ens udforskning af kortet. Når et vejkryds ses, vil dets sideveje og usikkerhed figurere i grafikvinduet. Dette opdateres løbende og indeholder også en statutstekst for hver aktiv SMR.

Måden hvorpå SMR'en udforsker miljøet er lavet, så den altid vælger udforskede sideveje. Findes der ikke udforskede sidevej i det vejkryds hvor den står, findes det nærmeste vejkryds med en eller flere udforskede sideveje. SMR'en kører derefter mod dette kryds. Når alle vejkryds er kortlagt – dvs. at der ikke er flere udforskede sideveje – vælger SMR'en de vejkryds, der er besøgt mindst antal gange. Derved opnås der at odometrifejl kan minimeres og en større præcision af kortet opnås.

Håndteringen af odometrifejl er foretaget ved tilskrivning af fejltolerancer på baggrund af kørt distance. Ved opdatering af data på kendte kryds, tages der højde for denne fejltolerance.

Der er foretaget en række test vha. af Visualizer til at illustrere kodens funktionalitet. Det er her vist hvordan en SMR udforsker et ukendt kort og hvordan kortets præcision forbedres ved fortsat kørsel. Der er også lavet en enkel demonstration af hvordan to SMR'er kan styres af programmet.

Opstart

For at køre med en SMR skal Visualizer, SMRDemo og Jess startes op. Følgende fremgangsmåde fra en terminal benyttes:

Mapperne kopieres fra Cd'en over på en computer.

Beskrivelse	Kommando
I ny Terminal - I mappen "smr" startes Visualizer:	<code>visualizer</code>
I ny Terminal - Start SMRDemo:	<code>smrdemo -t1 -s0</code>
I ny Terminal - I mappen 'Jess' startes Jess:	<code>jess</code>
Skriv derefter i Jess	<code>(batch go.jss)</code>
...efterfuldt af	<code>(run-until-halt)</code>

B Templates

For at kunne beskrive data på en hensigtsmæssig og overskuelig måde, benyttes en række templates til at angive kortets opbygning samt de data for SMR'en, der skal bruges til udregning af kortets veje og vejkryds.

B.1 Verticelist (Liste med alle vejkryds)

Den sidste template der defineres, er en liste der benyttes, når den korteste vej til et uudforsket vejkryds skal findes.

```
(deftemplate verticelist (slot robot) (multislot vertices))
```

robot Angiver hvilken SMR der er tale om

vertices Angiver hvilke Vertice's der undersøgt i sin søgen efter et Vertice med uudforsket LineEnds.

B.2 SmrVar (Hjælpevariable)

Denne templates beskriver hvilken SMR der skal benyttes i det tilfælde, at der køres med mere end en. Den indeholder desuden hjælpevariable som SMR længde, odometrifejlfaktor, sortlinje grænseværdi, kørt distance osv. Dermed kan disse værdier benyttes for en bestemt SMR, når der f.eks. skal udregnes koordinater for et nyt Vertice..

```
(deftemplate SmrVar (slot robot) (slot AllowTalk) (slot SmrLineTh) (slot SmrLength)
  (slot errFactor) (slot LastLineEnd) (slot LastDrivenDist) (slot State) (slot
  curVertice) (slot wantedDir) (multislot verticelists) (slot calcDir))
```

robot Slottet *robot* Angiver navnet på en specifik SMR i det tilfælde, at der køres med mere end en.

Allowtalk *Allowtalk* Kan være 'yes' eller 'no' og angiver, om det er tilladt at køre med den ønskede SMR.

SmrLineTh Dette er en grænseværdi for lyssensorerne, så det er muligt at detektere f.eks. en tværgående sort linje(sidevej).

SmrLenght Denne værdi er længden på SMR'en. Denne bruges når et koordinat for en Vertice skal udregnes. I det øjeblik en SMR ser et Vertice, kan dets rigtige position ikke udregnes kun ud fra odometri.

errFactor *errFactor* Er den fejlfaktor (fejl pr. kørt distance) som SMR'en har.

maxErr *maxErr* Er den fejlfaktor som maksimum kan antages, for ikke at fejlen bliver så stor at flere punkter regnes som værende et.

LastLineEnd Slottet angiver den sidste LineEnd der er kørt ad.

LastDrivenDist For at udregne en distance for en Line, benyttes *LastDrivenDist*, som indeholder den total kørte distance ved sidste Vertice.

State Dette angiver hvilken state SMR'en er i. Dette kan være Idle, DetectingSupportV eller Drive, og angiver om en SMR er klar til at en bestemt regel eksekveres på den.

curVertice Indeholder fact-adressen på det Vertice som en SMR står ved pt.

folLineEnd Angiver hvilken LineEnd en SMR skal følge, når det undersøges om et uudforsket LineEnd skal følges.

verticeLists Indeholder en liste af VerticeList. Se nedenstående beskrivelse i afsnit B.1.

calcDir Kan være "yes" eller "no" og benyttes som et flag der angiver, om der skal beregnes en ny drejeretning.

StatusText Bruges til at vise statutstekst i grafikvinduet.

B.3 Vertice (Vejkryds)

Et Vertice indeholder information om hvor vejkrydset ligger f. eks. 'x' og 'y' koordinater. Et Vertice ser ud som følgende:

```
(deftemplate Vertice (slot x) (slot y) (slot estErr) (slot status) (slot timesVisited) (
  slot SmrDiscovered))
```

x,y Slot *x* og *y* er som nævnt koordinatet på det givne Vertice.

estErr Dette er en fejl faktor, der angiver radius på den usikkerhed, der er for placering af et Vertice's. Dvs. et Vertice opfattes ikke kun som et punkt, men også med at areal i form af en cirkel der angiver usikkerheden.

Status Desuden har et Vertice også et status som kan antage fire følgende værdier: Supporter, Prospect, FullBlood og BadStanding.

timesVisited Slottet *timesVisited* angiver antallet at gange et Vertice et blevet undersøgt.

B.4 LineEnd (Sidevej)

En Line indeholder information om hvilket Vertice det er tilknyttet. Templates ser ud som følgende:

```
(deftemplate LineEnd (slot Angle) (slot Vertice))
```

Angle Indeholder vinkelretningen på et LineEnd, og er den retning som LineEnd'en peger i forhold til et Vertice.

Vertice Indeholder adressen på det Vertice som LineEnd'en er forbundet til.

B.5 Line (Vej)

En Line indeholder information om hvilke to *LineEnds* det er tilknyttet og en længde. Templates ser ud som følgende:

```
(deftemplate Line (multislot LineEnd) (slot Length))
```

LineEnd Indeholder adresserne på de to LineEnds der angiver endepunkterne for en Line.

C.1 robots.jss

```
1 (assert (useRobot
2   (name "nyquist")
3   (x 0)
4   (y 0)
5   (angle 0)
6 ))
7
8 ;(assert (useRobot
9   ; (name "nyquist:31002")
10  ; (x 2.5)
11  ; (y 3.7)
12  ; (angle 180)
13  ;))
```

C.2 go.jss

```
1
2 (reset)
3 (clear)
4
5 (load-package jessmr.pkg)
6
7 (deftemplate useRobot
8   (slot name)
9   (slot x)
10  (slot y)
11  (slot angle)
12 )
13
14 (defquery getRobots
15   ?addr<-(useRobot (name ?name) (x ?x) (y ?y) (angle ?angle))
16 )
17
18 (batch robots.jss)
19
20 (deffunction connect2Robots()
21   (bind ?robots (run-query* getRobots))
22   (while (?robots next)
23     (bind ?r (?robots getString name))
24     (bind ?pos (str-index ":" ?r))
25     (if (integerp ?pos) then
26       (bind ?name (sub-string 1 (- ?pos 1) ?r))
27       (bind ?port (sub-string (+ ?pos 1) (str-length ?r) ?r))
28       (SMRConnect ?name (integer ?port))
29     else
30       (SMRConnect ?r)
31   )
```

```

32  )
33  )
34
35  (connect2Robots)
36
37  (batch templates.jss)
38  (batch mapManager.jss)
39  (batch SmrComm.jss)
40  (batch gui.jss)
41
42  (deffunction addSmrVars()
43    (bind ?robots (run-query* getRobots))
44    (while (?robots next)
45      (bind ?r (?robots getString name))
46      (assert (SmrVar
47        (robot ?r)
48        (AllowTalk no)
49        (SmrLineTh 0.5)
50        (SmrLength 0.25)
51        (LastLineEnd nil)
52        (errFactor 0.01)
53        (maxErr 0.23)
54        (curVertice nil)
55        (calcDir no)
56        (State Init)
57        (StatusText "Initializing")
58      )))
59
60    ; The following commands ensures that the robot thinks it is where we have placed it
61    ; and makes sure the internal destination variables for angle, x and y is reset
62    (SMRTalk "idle" ?r)
63    (SMRTalk (format nil "set \"$odoth\" %f" (/ (* (?robots getFloat angle) (pi)) 180)) ?r
64      )
65    (SMRTalk (format nil "set \"$odox\" %f" (?robots getFloat x)) ?r)
66    (SMRTalk (format nil "set \"$odoy\" %f" (?robots getFloat y)) ?r)
67    (SMRTalk (format nil "turn %f" (?robots getFloat angle)) ?r)
68    (SMRTalk "drive : (1)" ?r)
69    (SMRTalk "idle" ?r)
70  )
71
72  (addSmrVars)
73
74  (deffunction cleanUpRobots()
75    (bind ?robots (run-query* getRobots))
76    (while (?robots next)
77      (retract (?robots get addr))
78    )
79    (return) ; without any return value
80  )
81
82  (cleanUpRobots)

```

C.3 templates.jss

```

1  (deftemplate verticeList
2    (slot robot)
3    (multislot vertices)
4  )
5
6  (deftemplate SmrVar
7    (slot robot)

```

```

8  (slot AllowTalk)
9  (slot SmrLineTh)
10 (slot SmrLength)
11 (slot errFactor)
12 (slot maxErr)
13 (slot LastLineEnd)
14 (slot LastDrivenDist)
15 (slot State)
16 (slot curVertice)
17 (slot folLineEnd)      ; LineEnd to follow out of a crossing
18 (multislot verticeLists) ; For calculating LineEnd to follow
19 (slot calcDir)
20 (slot StatusText)    ; Status text to display in graphic window
21 )
22
23
24 (deftemplate Vertice
25   (slot x)
26   (slot y)
27   (slot estErr)
28   (slot status)
29   (slot timesVisited)
30 )
31
32
33 (deftemplate LineEnd
34   (slot Angle)
35   (slot Vertice)
36 )
37
38
39 (deftemplate Line
40   (multislot LineEnd)
41 )

```

C.4 SmrComm.jss

```

1  ; ----- Functions -----
2  (deffunction CalcDirection (?theta)
3    (if (<= ?theta -5.498) then (return 0))
4    (if (and (> ?theta -5.498) (<= ?theta -3.927)) then (return 90))
5    (if (and (> ?theta -3.927) (<= ?theta -2.356)) then (return 180))
6    (if (and (> ?theta -2.356) (<= ?theta -0.785)) then (return 270))
7    (if (and (> ?theta -0.785) (<= ?theta 0.785)) then (return 0))
8    (if (and (> ?theta 0.785) (<= ?theta 2.356)) then (return 90))
9    (if (and (> ?theta 2.356) (<= ?theta 3.927)) then (return 180))
10   (if (and (> ?theta 3.927) (<= ?theta 5.498)) then (return 270))
11   (if (> ?theta 5.498) then (return 0))
12 )
13
14
15 (deffunction OffsetX (?x ?theta ?SmrLenght)
16   return (+ ?x (* (Math.cos ?theta) ?SmrLenght))
17 )
18
19
20 (deffunction OffsetY (?y ?theta ?SmrLenght)
21   return (+ ?y (* (Math.sin ?theta) ?SmrLenght))
22 )
23
24
25 (deffunction CalcEstErr (?fSmrVar ?DrivenDist)

```

```

26 (bind ?fLastLineEnd (fact-slot-value ?fSmrVar LastLineEnd))
27 (if (= (jess-type ?fLastLineEnd) FACT) then
28   (bind ?LastEstErr (fact-slot-value
29     (fact-slot-value
30       (fact-slot-value ?fSmrVar LastLineEnd)
31       Vertice)
32     estErr)
33   )
34 else
35   (bind ?LastEstErr 0.0)
36 )
37 (bind ?NewEstErr (sqrt (+ (** ?LastEstErr 2) (* (- ?DrivenDist (fact-slot-value ?fSmrVar
38   LastDrivenDist)) (fact-slot-value ?fSmrVar errFactor))))))
39 (return (min (fact-slot-value ?fSmrVar maxErr) ?NewEstErr))
40 )
41
42
43 (deffunction SMRReset(?SmrVar)
44   (SMRTalk "flushcmds" (fact-slot-value ?SmrVar robot))
45   (SMRTalk (format nil "SMRth=%f" (fact-slot-value ?SmrVar SmrLineTh)) (fact-slot-value ?
46     SmrVar robot))
47   (SMRTalk (format nil "SMRrobotlength=%f" (fact-slot-value ?SmrVar SmrLength)) (fact-slot
48     -value ?SmrVar robot))
49   (SMRTalk "idle" (fact-slot-value ?SmrVar robot)) ; ekstra dummy cmd to ensure SMRDemo
50     parses the vars...
51 )
52
53 (deffunction SMRTalk2(?cmd ?SmrVar)
54   (modify ?SmrVar (AllowTalk no))
55   (SMRTalk ?cmd (fact-slot-value ?SmrVar robot))
56 )
57
58 (deffunction SMRcheckForIntersection()
59   (format nil "($line0 < SMRth & $line1 < SMRth) | ($line6 < SMRth & $line7 < SMRth)")
60 )
61
62 (deffunction SMRcheckForNoLine()
63   (format nil "($line0 > SMRth & $line1 > SMRth & $line2 > SMRth & $line3 > SMRth & $line4
64     > SMRth & $line5 > SMRth & $line6 > SMRth & $line7 > SMRth)")
65 )
66
67 ; ----- Rules -----
68
69 (defrule Init
70   ?SmrVar<-(SmrVar (robot ?r) (State Init))
71   (odometry (robot ?r) (x ?x) (y ?y) (theta ?theta) (dist ?dist))
72   =>
73   (SMRReset ?SmrVar)
74   (SMRTalk2 (format nil "followline bm : (%s) | (%s)" (SMRcheckForIntersection) (
75     SMRcheckForNoLine)) ?SmrVar)
76   (SMRTalk2 "idle" ?SmrVar)
77   (modify ?SmrVar (LastDrivenDist 0) (State Drive) (StatusText "Searching for first
78     vertice"))
79 )
80
81 (defrule SMRTalkAllowed ;Are we allowed to send commands ?
82   ?SmrVar<-(SmrVar (robot ?r) (AllowTalk no))
83   (currentcommand (status "done 0") (id ?id) (robot ?r))
84   (commandqueue (id ?id) (robot ?r))

```

```

85 =>
86 (modify ?SmrVar (AllowTalk yes) (calcDir yes))
87 )
88
89
90
91 (defrule DetectSupportVertice (declare (salience 10))
92   ?fSmrVar<-(SmrVar (robot ?r) (AllowTalk yes) (SmrLineTh ?SmrLineTh) (LastLineEnd ?
93     fLineEndLast) (SmrLength ?SmrLength) (State Drive))
94   (test (or
95     (and (< ?l0 ?SmrLineTh) (< ?l1 ?SmrLineTh))
96     (and (< ?l6 ?SmrLineTh) (< ?l7 ?SmrLineTh))
97     (and (> ?l0 ?SmrLineTh) (> ?l1 ?SmrLineTh) (> ?l2 ?SmrLineTh) (> ?l3 ?SmrLineTh) (> ?
98       l4 ?SmrLineTh) (> ?l5 ?SmrLineTh) (> ?l6 ?SmrLineTh) (> ?l7 ?SmrLineTh))
99   ))
100   (odometry (robot ?r) (x ?x) (y ?y) (theta ?theta) (dist ?dist))
101 =>
102   ;Opretter nyt vertice
103   (bind ?fVerticeNew (assert (Vertice (x (OffsetX ?x ?theta ?SmrLength)) (y (OffsetY ?y ?
104     theta ?SmrLength)) (estErr (CalcEstErr ?fSmrVar ?dist)) (timesVisited 1) (status
105     Supporter))))
106   ;Tilføjer LineEnds til vertice
107   (bind ?fLineEndNewB (assert (LineEnd (Vertice ?fVerticeNew) (Angle (CalcDirection (- ?
108     theta (pi)))))))
109   (if (and (< ?l0 ?SmrLineTh) (< ?l1 ?SmrLineTh)) then
110     (bind ?fLineEndNewR (assert (LineEnd (Vertice ?fVerticeNew) (Angle (CalcDirection (- ?
111       theta (/ (pi) 2)))))))
112   )
113   (if (and (< ?l6 ?SmrLineTh) (< ?l7 ?SmrLineTh)) then
114     (bind ?fLineEndNewL (assert (LineEnd (Vertice ?fVerticeNew) (Angle (CalcDirection (+ ?
115       theta (/ (pi) 2)))))))
116   )
117   ;Tilføjer forbinder LineEnds med Line
118   (if (= (jess-type ?fLineEndLast) FACT) then
119     (bind ?fVerticeLast (fact-slot-value ?fLineEndLast Vertice))
120     (bind ?fLine (assert (Line (LineEnd ?fLineEndLast ?fLineEndNewB))))
121   )
122   (modify ?fSmrVar (LastLineEnd ?fLineEndNewB) (State DetectingSupportV))
123   (SMRTalk2 (format nil "fwd %f" ?SmrLength) ?fSmrVar)
124 )
125 )
126
127
128
129
130
131 (defrule DetectLastLineEnd (declare (salience 10))
132   ?fSmrVar<-(SmrVar (robot ?r) (AllowTalk yes) (SmrLineTh ?SmrLineTh) (LastLineEnd ?
133     fLineEndLast) (State DetectingSupportV))
134   (linesensor (robot ?r) (values $?l))
135   (odometry (robot ?r) (theta ?theta))
136 =>
137   (bind ?fVerticeLast (fact-slot-value ?fLineEndLast Vertice))
138   (bind ?lmin (min (nth$ 3 ?l) (nth$ 4 ?l) (nth$ 5 ?l) (nth$ 6 ?l)))
139   (if (< ?lmin ?SmrLineTh) then
140     (bind ?fLineEndNew (assert (LineEnd (Vertice ?fVerticeLast) (Angle (CalcDirection ?
141       theta))))))
142   (modify ?fSmrVar (State Idle))

```

```

142 (modify ?fVerticeLast (status Prospect))
143 )
144
145
146 (defrule Drive (declare (salience -100))
147   ?SmrVar<-(SmrVar (robot ?r) (AllowTalk yes) (State Idle) (folLineEnd ?followLE) (
148     curVertice ?curV))
149   (odometry (robot ?r) (theta ?theta) (dist ?dist))
150   =>
151   (bind ?ang (CalcDirection ?theta))
152   (bind ?wantedAng (fact-slot-value ?followLE Angle))
153   (bind ?turn (- ?wantedAng ?ang))
154
155   ; Setting current robot position to current vertice coordinates
156   (SMRTalk2 "idle" ?SmrVar) ; Make sure position control is disable in SMRDemo
157   (SMRTalk2 (format nil "set \"$odox\" %f" (fact-slot-value ?curV x)) ?SmrVar) ; The
158     command following set-commands shoul be a drive command
159   (SMRTalk2 (format nil "set \"$odoy\" %f" (fact-slot-value ?curV y)) ?SmrVar) ; E.g. not
160     a fwd command
161
162   (SMRTalk2 (format nil "turn %d" ?turn) ?SmrVar)
163   (SMRTalk2 (format nil "followline bm : (%s) | (%s)" (SMRcheckForIntersection) (
164     SMRcheckForNoLine)) ?SmrVar)
165   (SMRTalk2 "idle" ?SmrVar)
166
167   (modify ?SmrVar (LastLineEnd ?followLE) (LastDrivenDist ?dist) (State Drive))
168 )
169
170 (deffunction cleanUpList(?vars)
171   (foreach ?element (fact-slot-value ?vars verticeLists) (retract ?element))
172   (modify ?vars (verticeLists))
173 )
174
175 (defquery lineBackChaining (declare (variables ?vertSrc ?list ?robot))
176   ?LEsrc<-(LineEnd (Vertice ?vertSrc)) ; Find the lineend containing the
177     vertice.
178   ?line<-(Line (LineEnd $? ?LEsrc $?)) ; Find the line connected to that
179     lineend
180   ?LEdst<-(LineEnd (Vertice ?vertDst)) ; Find another lineend
181   ?line<-(Line (LineEnd $? ?LEdst $?)) ; Also connected to that line
182   ?list<-(verticeList (robot ?robot) (vertices $? ?vertDst $?)) ; The dst vertice should
183     be in the given list
184 )
185
186 (deffunction calcDirToDstVertice(?smr ?dst)
187   (bind ?src (fact-slot-value ?smr curVertice))
188   (bind ?r (fact-slot-value ?smr robot))
189   (bind ?lists (fact-slot-value ?smr verticeLists))
190   (bind ?i (- (length$ ?lists) 1))
191   (while (> ?i 0)
192     (bind ?list (nth$ ?i ?lists))
193     (bind ?result (run-query* lineBackChaining ?dst ?list ?r))
194     (if (?result next) then ; There should be one and only one match
195       (bind ?dst (?result get vertDst))
196       (bind ?list (?result get list))
197     )
198     (if (= ?src ?dst) then ; Found back to where we are standing
199       (return (?result get LEdst)) ; We are done. Return desired LineEnd
200     )

```

```

201     )
202     else
203         (return nil)
204     )
205     (bind ?i (- ?i 1))
206 )
207
208 (return nil)
209 )
210
211 (defrule posUpdate (declare (salience 5))
212     ?SmrVar<-(SmrVar (robot ?r) (curVertice ?cv) (LastLineEnd ?lastLE))
213     (test (= (jess-type ?lastLE) FACT))
214     (odometry (robot ?r) (x ?xr) (y ?yr) (dist ?dist))
215
216     ?v1<-(Vertice (status FullBlood) (x ?x1) (y ?y1) (estErr ?estErr1)) ; Find a
        fullblood
217
218     (not (and ; It is not possible to
219         (Vertice (status FullBlood) (x ?x2) (y ?y2)) ; Find another
        fullblood
220         (test (< (calcDist ?xr ?yr ?x2 ?y2) (calcDist ?xr ?yr ?x1 ?y1)) ; which is
        closer to us.
221     ))
222
223     (test (< (calcDist ?xr ?yr ?x1 ?y1) (+ (CalcEstErr ?SmrVar ?dist) ?estErr1))) ;
        Distance should be smaller than err
224
225     (test (or
226         (<> (jess-type ?cv) FACT) ; Either should the curVertice (cv
        ) not be a fact (== nil), or...
227         (<> ?v1 ?cv) ; the found vertice (v) should not be
        equal to cv
228     ))
229 =>
230     (modify ?SmrVar (curVertice ?v1) (calcDir yes))
231 )
232
233 (defquery findLostLineEnds (declare (variables ?vertice))
234     ?LE<-(LineEnd (Vertice ?vertice)) ; A lineend associated to ?vertice is found...
235     (not (Line (LineEnd $? ?LE $?))) ; that is not associated with a line
236 )
237
238 (defquery findConnectedVertices (declare (variables ?robot ?includeList))
239     ?includeList<-(verticeList (vertices $? ?vertSrc ??)) ; 1. Find a vertice in the
        include list...
240     ?lineendSrc<-(LineEnd (Vertice ?vertSrc)) ; and its lineend...
241
242     ?lineendDst<-(LineEnd (Vertice ?vertDst)) ; 2. Find another lineend and
        vertice...
243     (not (verticeList (robot ?robot) (vertices $? ?vertDst $?))) ; which is not
        included in a verticeList
244     ; ^^^ (In other words - we must not have examined
        that vertice before)
245     ?vertDst<-(Vertice (status FullBlood) (timesVisited ?timesVisited)) ; but has status
        Fullblood
246
247
248     ?line<-(Line (LineEnd $? ?lineendSrc $?)) ; and a line connected from 1.
        ...
249     ?line<-(Line (LineEnd $? ?lineendDst $?)) ; to 2. ...
250 )
251
252 (defquery findMinvisited (declare (variables ?excludeVertice))
253     ?v<-(Vertice (timesVisited ?minVisit) (status FullBlood)) ; Find a vertice...
254     (not (and ; with a minimum value in timesVisited

```

```

255     (Vertice (timesVisited ?visit))
256     (test (< ?visit ?minVisit))
257   ))
258
259   (test (<> ?excludeVertice ?v))           ; That is not the ecluded vertice (usually
      the one we are standing in..)
260 )
261
262
263 (defrule updWantedDir1 "Checks if the current vertice has unexplored lineends"
264   ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists) (State
      Idle)) ; An update is needed and we have not checked any vertices yet
265   (test (= (jess-type ?curVert) FACT))
266   =>
267   (bind ?result (run-query* findLostLineEnds ?curVert))           ; Search for
      lost vertices
268
269   (if (?result next) then                                           ; If a vertice is found
270     (modify ?SmrVar (folLineEnd (?result get LE)) (calcDir no)     ; We'
      re done... easy
271     (StatusText (str-cat "Leaving " (fact-slot-value (?result get LE) Vertice) " via "
      (?result get LE))))
272   else                                                               ; else
273     (modify ?SmrVar (verticeLists (assert (verticeList (robot ?r) (vertices ?curVert))))
      ) ; Add current vertice to list.
274   )
275 )
276
277 (defrule updWantedDir2 "Checks if connected vertices (in any depth) has unexplored
      lineends"
278   ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists $? ?
      lastList) (State Idle)) ; An update is needed and we have checked
279   ?fLineEnd<-(LineEnd (Vertice ?vertice))                           ; There is
      at least one lineend...
280   (not (Line (LineEnd $? ?fLineEnd $?)))                             ; not
      associated to a line...
281   =>
282   (bind ?newList (assert (verticeList (robot ?r))))                 ; Create
      entry to visited vertice list
283   (modify ?SmrVar (verticeLists (fact-slot-value ?SmrVar verticeLists) ?newList))
      ; add verticeList to smrvar
284
285   (bind ?cont 1)                                                   ; Used for break
      condition
286   (bind ?vertices (run-query* findConnectedVertices ?r ?lastList))
      ; Find connected vertices
287   (while (and (?vertices next) (= ?cont 1))                         ; For each
      of ^^^.
288     (bind ?ends (run-query* findLostLineEnds (?vertices get vertDst)))
      ; check if one of them has a loose end.
289     (if (?ends next) then
290       (modify ?SmrVar (folLineEnd (calcDirToDstVertice ?SmrVar (?vertices get vertDst)))
      (calcDir no) ; calculate direction for our trip
291       (StatusText (str-cat "Heading for " (?vertices get vertDst))))
292       (cleanUpList ?SmrVar) ; clean up after our
      selves.
293       (bind ?cont 0) ; break the loop
294     else ; else...
295       (modify ?newList (vertices (fact-slot-value ?newList vertices) (?vertices get
      vertDst))) ; Add vertice to verticeList
296   )
297 )
298 )
299
300 (defrule updWantedDir3 "If no unexplored lineends exists => Goto vertice with min
      timesvisited value"

```

```

301     ?SmrVar<-(SmrVar (robot ?r) (calcDir yes) (curVertice ?curVert) (verticeLists $? ?
        lastList) (State Idle)) ; An update is needed and we have checked
302     (not (and                                     ; Invert:
303         ?fLineEnd<-(LineEnd (Vertice ?vertice))
                                     ; There
        is at least one lineend...
304         (not (Line (LineEnd $? ?fLineEnd $?)))
                                     ; not
        associated to a line...
305     ))
306 =>
307     (bind ?minVisitList (run-query* findMinvisited ?curVert))
        ; a minimum of times
308     (if (?minVisitList next) then
        ; There should at
        least be one vertice
309         (bind ?minVisitedValue (?minVisitList getInt minVisit))
        ;
        Remember the value
310     else
        ; else (if this happens, a bug
        has been discovered)
311         (halt)
        ; Play it safe...
312     )
313 )
314
315     (bind ?newList (assert (verticeList (robot ?r)))
        ; Create
        entry to visited vertice list
316     (modify ?SmrVar (verticeLists (fact-slot-value ?SmrVar verticeLists) ?newList))
        ; add verticeList to smrvar
317
318     (bind ?cont 1)
        ; Used for break
        condition
319     (bind ?vertices (run-query* findConnectedVertices ?r ?lastList))
        ; Find connected vertices
320     (while (and (?vertices next) (= ?cont 1))
        ; For each
        of ^^^.
321         (if (= ?minVisitedValue (?vertices getInt timesVisited)) then
322             (modify ?SmrVar (folLineEnd (calcDirToDstVertice ?SmrVar (?vertices get vertDst))
                (calcDir no) ; calculate direction for our trip
                (StatusText (str-cat "Heading for " (?vertices get vertDst) " (least visited)"))
            )
323             (cleanUpList ?SmrVar)
        ; clean up after our
        selves.
324         (bind ?cont 0)
        ; break the loop
325     else
        ; else...
326         (modify ?newList (vertices (fact-slot-value ?newList vertices) (?vertices get
            vertDst)))
        ; Add vertice to verticeList
327     )
328 )
329 )
330 )

```

C.5 MapManager.jss

```

1
2 ; ----- Functions -----
3
4 (deffunction calcNewEstErr (?estErrNew ?estErrOld ?n)
5     (return (/
6         (+ ?estErrNew (* ?n ?estErrOld))
7         (+ ?n 1)
8     )
9 )
10 )
11
12 (deffunction calcDist (?x1 ?y1 ?x2 ?y2)
13     (return (sqrt (+

```

```

14      (** (- ?x1 ?x2) 2)
15      (** (- ?y1 ?y2) 2)
16    ))
17  )
18 )
19
20 (deffunction calcCoordinate (?cNew ?errNew ?cOld ?errOld ?nOld)
21   (return
22     (/
23       (+ (* ?nOld ?errNew ?cOld) (* ?cNew ?errOld))
24       (+ (* ?nOld ?errNew) ?errOld)
25     )
26   )
27 )
28
29
30 ; ----- Rules -----
31
32
33 (defrule MergeVertice
34   ?fProspect<- (Vertice (status Prospect) (x ?x1) (y ?y1) (estErr ?estErr1))
35   ?fLineEndP<- (LineEnd (Vertice ?fProspect) (Angle ?Angle))
36   ?fLineP<- (Line (LineEnd $?fLineEndPrev ?fLineEndP $?fLineEndAfter))
37   ?fFullBlood<- (Vertice (status FullBlood) (x ?x2) (y ?y2) (estErr ?estErr2) (timesVisited
38     ?n))
39   (test (< (calcDist ?x1 ?y1 ?x2 ?y2) (+ ?estErr1 ?estErr2)))
40   ?fLineEndF<- (LineEnd (Vertice ?fFullBlood) (Angle ?Angle))
41   =>
42   (bind ?errFromDist (/ (calcDist ?x1 ?y1 ?x2 ?y2) 2))
43
44   (modify ?fLineP (LineEnd $?fLineEndPrev ?fLineEndF $?fLineEndAfter))
45   (modify ?fFullBlood
46     (x (calcCoordinate ?x1 ?errFromDist ?x2 ?estErr2 ?n))
47     (y (calcCoordinate ?y1 ?errFromDist ?y2 ?estErr2 ?n))
48     (estErr (calcNewEstErr ?errFromDist ?estErr2 ?n))
49     (timesVisited (+ ?n 1))
50   )
51   (modify ?fProspect (status BadStanding))
52 )
53
54 (defrule PromoteVertice
55   ?f1<- (Vertice (status Prospect) (x ?x1) (y ?y1) (estErr ?estErr1))
56   (not (and
57     (Vertice (status FullBlood) (x ?x2) (y ?y2) (estErr ?estErr2))
58     (test (< (calcDist ?x1 ?y1 ?x2 ?y2) (+ ?estErr1 ?estErr2)))
59   ))
60   =>
61   (modify ?f1 (status FullBlood) (timesVisited 1))
62 )
63
64
65
66 (defrule DeleteLineEnd
67   ?fVertice<- (Vertice (status BadStanding))
68   ?fLineEnd<- (LineEnd (Vertice ?fVertice))
69   =>
70   (retract ?fLineEnd)
71 )
72
73
74
75 (defrule DeleteVertice
76   ?fVertice<- (Vertice (status BadStanding))
77   (not (LineEnd (Vertice ?fVertice)))
78   =>

```

```

79     (retract ?fVertice)
80
81 )
82
83
84 (defrule DeleteDuplicateLines1
85   ?fLine1<- (Line (LineEnd ?fLineEnd1 ?fLineEnd2))
86   ?fLine2<- (Line (LineEnd ?fLineEnd2 ?fLineEnd1))
87   (test (<> (fact-id ?fLine1) (fact-id ?fLine2)))
88   =>
89     (retract ?fLine2)
90 )
91
92
93 (defrule DeleteDuplicateLines2
94   ?fLine1<- (Line (LineEnd ?fLineEnd1 ?fLineEnd2))
95   ?fLine2<- (Line (LineEnd ?fLineEnd1 ?fLineEnd2))
96   (test (<> (fact-id ?fLine1) (fact-id ?fLine2)))
97   =>
98     (retract ?fLine2)
99 )

```

C.6 gui.jss

```

1  (import javax.swing.*)
2  (import javax.swing.JFrame)
3  (import java.awt.*)
4  (import java.awt.event.*)
5  (import java.awt.Color)
6  (import java.awt.BorderLayout)
7
8
9
10 ; Set up 3 buttons to start and stop
11 (deffunction setupControl()
12   (bind ?frame (new JFrame "Robot Control Center - a.k.a. Rouston"))
13   (bind ?panel (new JPanel (new GridLayout 0 2)))
14
15   (bind ?bHalt (new JButton "Halt"))
16   (?panel add ?bHalt (BorderLayout.CENTER))
17
18   (bind ?bToggleID (new JButton "Toggle IDs"))
19   (?panel add ?bToggleID (BorderLayout.CENTER))
20
21   (bind ?pane (get ?frame contentPane))
22   (?pane add ?panel (BorderLayout.CENTER))
23
24   (?bHalt addActionListener (implement ActionListener using
25     (lambda (?name ?event)
26       (halt)
27       (printout t "Halt" crlf))))
28
29   (?bToggleID addActionListener (implement ActionListener using
30     (lambda (?name ?event)
31       (bind ?*graphicToggleID* (- 1 ?*graphicToggleID*))
32       (if (java-objectp ?*graphicRootPanel*) then (?*graphicRootPanel* repaint))
33     )))
34
35   (?frame pack)
36   (set ?frame visible TRUE)
37   (?frame setDefaultCloseOperation (WindowConstants.DISPOSE_ON_CLOSE))
38 )

```

```

39
40
41
42
43
44 ;;; Set up the drawing...
45 (deffunction painter (?canvas ?graph)
46   (bind ?res 100)                ; Resolution pixels pr. m.
47   (bind ?border 0.40)           ; map border in meters
48   (bind ?lineendlength 0.10)   ; Length of lineends in metres
49   (bind ?textHeight 20)
50   (bind ?minWinSize 150)
51
52
53   ; find max and min x and y coordinates, calc offset and set window size
54   (bind ?minmax (run-query* findMinMaxCoordinates))
55   (if (?minmax next) then
56     (bind ?minx (?minmax getFloat xmin))
57     (bind ?maxx (?minmax getFloat xmax))
58     (bind ?miny (* -1 (?minmax getFloat ymax)))
59     (bind ?maxy (* -1 (?minmax getFloat ymin)))
60   else
61     (bind ?maxx 0)
62     (bind ?maxy 0)
63     (bind ?minx 0)
64     (bind ?miny 0)
65   )
66   (bind ?offsetx (- ?border ?minx))
67   (bind ?offsety (- ?border ?miny))
68
69   (bind ?offsety (+ ?offsety (* (/ ?textHeight ?res) (+ 1 (count-query-results getSmrVars)
70     ))) ; Make room for text at top
71
72   (bind ?width (get-member (?canvas getSize) width))
73   (bind ?height (get-member (?canvas getSize) height))
74   (bind ?newWidth (max (round (* ?res (+ ?border ?maxx ?offsetx))) (* 2 ?minWinSize)))
75   (bind ?newHeight (max (round (* ?res (+ ?border ?maxy ?offsety))) ?minWinSize))
76   (if (or (<> ?newHeight ?height) (<> ?newWidth ?width)) then
77     (bind ?diffW (- (get-member (?*graphicWindow* getSize) width) ?width))
78     (bind ?diffH (- (get-member (?*graphicWindow* getSize) height) ?height))
79     (?*graphicWindow* setSize (+ ?newWidth ?diffW) (+ ?newHeight ?diffH))
80   )
81
82
83   ; Clear window
84   (?graph setColor (Color.black))
85   (?graph fillRect 0 0 ?width ?height)
86
87
88   ; Write status text
89   (?graph setColor (Color.blue))
90   (bind ?smrs (run-query* getSmrVars))
91   (bind ?x (round (/ ?textHeight 2)))
92   (bind ?y (round (/ ?textHeight 2)))
93   (while (?smrs next)
94     (?graph drawString (str-cat (?smrs getString robot) ": " (?smrs getString StatusText))
95       ?x ?y)
96     (bind ?y (+ ?y ?textHeight))
97   )
98
99   ; Draw lines
100  (?graph setColor (Color.white))
101  (bind ?lines (run-query* findLines))
102  (while (?lines next)

```

```

103     (bind ?from (fact-slot-value (?lines get LineEnd1) Vertice))
104     (bind ?to (fact-slot-value (?lines get LineEnd2) Vertice))
105
106     (bind ?fromx (round (* ?res (+ (fact-slot-value ?from x) ?offsetx))))
107     (bind ?fromy (round (* ?res (+ (* (fact-slot-value ?from y) -1) ?offsety))))
108     (bind ?tox (round (* ?res (+ (fact-slot-value ?to x) ?offsetx))))
109     (bind ?toy (round (* ?res (+ (* (fact-slot-value ?to y) -1) ?offsety))))
110
111     (?graph drawLine ?fromx ?fromy ?tox ?toy)
112     )
113
114
115
116 ; Draw LineEnds
117 (?graph setColor (Color.green))
118     (bind ?ends (run-query* findLineEnds))
119     (while (?ends next)
120
121         (bind ?fromx (round (* ?res (+ ?offsetx (?ends getFloat x))))))
122         (bind ?fromy (round (* ?res (+ ?offsety (* (?ends getFloat y) -1))))))
123
124         (bind ?angle (* (?ends getFloat angle) (pi) 0.0055555556))
125         (bind ?tox (round (* ?res (+ ?offsetx (?ends getFloat x) (* (Math.cos ?angle) ?
126             lineendlength))))))
127         (bind ?toy (round (* ?res (+ ?offsety (* (?ends getFloat y) -1) (* -1 (Math.sin ?angle)
128             ?lineendlength))))))
129
130     (?graph drawLine ?fromx ?fromy ?tox ?toy)
131     )
132
133
134 ; Draw vertices - circles to illustrate their error
135 (?graph setColor (Color.red))
136 (bind ?vertices (run-query* findVertices))
137 (while (?vertices next)
138     (bind ?err (round (* 2 ?res (?vertices getFloat estErr))))
139     (bind ?x (round (+ (* -0.5 ?err) (* ?res (+ ?offsetx (?vertices getFloat x))))))
140     (bind ?y (round (+ (* -0.5 ?err) (* ?res (+ ?offsety (* (?vertices getFloat y) -1))))))
141     )
142     (?graph drawOval ?x ?y ?err ?err)
143
144     (if (= ?*graphicToggleID* 1) then
145         (bind ?textoffset (round (* ?res (?vertices getFloat estErr))))
146         (bind ?text (str-cat (?vertices get id)))
147         (bind ?text (sub-string 7 (- (str-length ?text) 1) ?text))
148         (bind ?text (str-cat ?text " (" (?vertices getInt timesVisited) ")"))
149         (?graph drawString ?text (- ?x ?textoffset 20) (- ?y ?textoffset 0))
150     )
151 )
152 )
153 )
154
155 (defglobal ?*graphicWindow* = 0)
156 (defglobal ?*graphicRootPanel* = 0)
157 (defglobal ?*graphicToggleID* = 1)
158 (deffunction setupPainterWindow()
159     (bind ?*graphicWindow* (new JFrame "Map"))
160     (?*graphicWindow* setDefaultCloseOperation (WindowConstants.DISPOSE_ON_CLOSE))
161     (bind ?*graphicRootPanel* (new jess.swing.JPanel painter (engine)))
162     ((?*graphicWindow* getContentPane) add ?*graphicRootPanel* (BorderLayout.CENTER))
163     (?*graphicWindow* setSize 50 50)
164     (?*graphicWindow* setVisible TRUE)
165 )

```

```

166
167
168
169 ;; Helper queueries...
170 (defquery findVertices
171   ?id<-(Vertice (x ?x) (y ?y) (estErr ?estErr) (timesVisited ?timesVisited))
172 )
173
174 (defquery findMinMaxCoordinates
175   (Vertice (x ?xmin))
176   (not (and
177     (Vertice (x ?x))
178     (test (< ?x ?xmin))
179   ))
180
181   (Vertice (x ?xmax))
182   (not (and
183     (Vertice (x ?x))
184     (test (> ?x ?xmax))
185   ))
186
187   (Vertice (y ?ymin))
188   (not (and
189     (Vertice (y ?y))
190     (test (< ?y ?ymin))
191   ))
192
193   (Vertice (y ?ymax))
194   (not (and
195     (Vertice (y ?y))
196     (test (> ?y ?ymax))
197   ))
198 )
199
200 (defquery findLineEnds
201   ?vertice<-(Vertice (x ?x) (y ?y))
202   (LineEnd (Angle ?angle) (Vertice ?vertice))
203 )
204
205 (defquery findLines
206   (Line (LineEnd ?LineEnd1 ?LineEnd2))
207   ?LineEnd1<-(LineEnd (Vertice ?V1))
208   ?V1<-(Vertice (status FullBlood))
209   ?LineEnd2<-(LineEnd (Vertice ?V2))
210   ?V2<-(Vertice (status FullBlood))
211 )
212
213 (defquery getSmrVars
214   (SmrVar (robot ?robot) (StatusText ?StatusText))
215 )
216
217
218 ;; Helper rule
219 (defrule graphicUpdate
220   (SmrVar)
221   =>
222     (if (java-objectp ?*graphicRootPanel*) then (?*graphicRootPanel* repaint))
223 )
224
225
226
227 ;; Starter
228 (deffunction setupGui()
229   (setupControl)
230   (setupPainterWindow)
231 )

```

```
232
233 ; http://herzberg.ca.sandia.gov/jess/docs/70/jessgui.html
234 ; http://java.sun.com/j2se/1.3/docs/api/index.html
235
236 (setupGui)
```