Förstudie

Reglering och kartering för en autonom robot

Lage Ragnarsson, Erik Sköld

Version 1.2

Status

Granskad		
Godkänd		

Sammanfattning

Denna förstudie är skriven som en del i kursen *TSEA56 - Kandidatprojekt i elektronik* vid Linköpings Universitet. I kursen ska en projektgrupp om sex personer konstruera en robot som autonomt ska kunna utforska och navigera i en labyrint.

De delar som behandlas i denna förstudie handlar om autonom körning, reglering av robotens rörelse samt kartering och kortaste vägen-problem.

Som lösning till robotens autonoma uppförande föreslås en tillståndsmaskin där det autonoma beteendet har brutits ner i mindre deluppgifter. Varje deluppgift har ett eget tillstånd och vid vissa beslutspunkter kan roboten byta tillstånd. Ett exempel på detta kan vara att gå från tillståndet att köra i en korridor till att köra utan reglering då roboten kommer fram till en korsning i labyrinten.

Under *reglering* behandlas hur robotens sensorer ska användas för att reglera körningen i labyrinten. Målet är en rak körning i korridorernas mitt. En PD-regulator föreslås men det diskuteras även om vår sensorkonfiguration gör att en kombination av två P-regulatorer skulle ha ett liknande beteende.

En lämplig kartrepresentation med tanke på omgivningens utseende presenteras och även hur data för kortaste vägen-beräkningar kan lagras i samma datastruktur som kartan. En vanlig algoritm för kortaste vägen-beräkningar, flood fill, presenteras och dess lämplighet diskuteras. Även små modifikationer till denna och en metod för att söka av labyrinten tas upp.

Innehåll

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2015-03-05	Tidigt utkast	LR, ESk	
1.0	2015-04-01	Första versionen	LR, ESk	
1.1	2015-04-23	Detaljer justerade efter återkoppling från handledare	LR, ESk	
1.2	2015-04-23	Figur 2 förtydligad och ekvation flyttad efter återkoppling från handledare	LR, ESk	

1 Inledning

Denna förstudie är en del av kursen *TSEA56 - Kandidatprojekt i elektronik* vid Linköpings universitet. Kursen bedrivs som ett projekt där grupper ska designa och bygga robotar för att utföra en specifik uppgift. Problemformuleringar och förslag till lösningar presenteras i relation till robotens uppdrag att utforska och navigera en labyrint. Förstudiens innehåll ligger till grund för de designbeslut som presenteras i designspecifikationen gällande styrmodulen.

1.1 Bakgrund

Gruppen ska konstruera en robot som autonomt ska utforska en labyrint och leta efter *nödställda* i denna samtidigt som den skapar en karta över omgivningen. Efter att roboten har hittat de nödställda ska den försäkra sig om att kortaste vägen från starten till de nödställda är känd och därefter återvända till starten för att plocka upp förnödenheter och leverera dessa. Detta uppdrag kan delas in i två deluppdrag, kartering och undsättning. Under karteringsfasen ska roboten utforska omgivningen och leta efter de nödställda, som egentligen är en speciellt markerad plats i labyrinten. Denna fas pågår tills dess att roboten är säker på att den vet kortaste vägen mellan starten och de nödställda. Den behöver alltså inte nödvändigtvis utforska hela labyrinten. Under undsättning ska roboten leverera en tom MER-förpackning från starten till de nödställda. Alla moment i uppdraget går på tid och en vinnare bland de grupper som gör samma projekt kommer utses enligt ett antal regler [1].

För att en robot ska kunna uppträda autonomt måste den bli medveten om sin omgivning och kunna ta beslut utifrån den insamlade informationen. Det autonoma elementet i roboten behöver därmed ersätta den funktion som en människa har i en användarstyrd robot.

Roboten behöver genom utforskning skapa sig en bild av omvärldens utformning och utifrån denna ta beslut om hur den ska navigera. När denna överblick skapats och utvärderats ska roboten kontinuerligt säkerställa att den önskade vägen följs genom att mäta avstånd och vinklar till väggar och reglera därefter. Under själva utforskningen behöver roboten dessutom aktivt ta beslut om vilket område den ska utforska och när utforskningen är klar.

1.2 Frågeställning och syfte

Denna förstudie har tre delar; *Tillstånd*, *Reglering* samt *Kartering och kortaste vägen*. Även om dessa relaterar till varandra vill vi separera dem i frågeställningen.

- *Hur kan autonomt beslutsfattande och autonom styrning realiseras?*
  Hur kan ett autonomt beteende struktureras och realiseras med en mikrokontroller så att roboten kan ta rätt beslut för att lösa sin uppgift?

- *Hur kan ett reglersystem konstrueras som håller roboten i korridorens mitt?*
  För att kunna orientera sig och navigera behöver roboten kunna köra rakt i korridorer, hur kan ett återkopplat reglersystem konstrueras för att åstadkommas detta?

- *Vilka kända algoritmer finns för utforskning av en labyrint?*
  Här studerar vi speciellt en algoritm som används för robottävlingsgrenen micromouse som har många likheter med vårt uppdrag. Hur fungerar dessa algoritmer i stort och lämpar de sig väl för vår tillämpning? Kan man modifiera någon så att den passar bättre?

Förstudiens syfte är att ge kunskap om dessa områden och fungera som ett beslutsstöd i projektarbetet. De metoder som presenteras ska ses som möjliga tillvägagångssätt och kommer inte nödvändigtvis att användas i projektet.

1.3 Avgränsningar

Denna förstudie har inte ambitionen att vara heltäckande på ämnena den berör. Var och ett av amnena är stora i sig och vi kommer bara ta upp det som är lämpligt att applicera på den robot gruppen ska konstruera.

Detta projekt kommer att genomföras med 8-bitars AVR-mikrokontroller. Vi planerar att använda en ATmega1284P som har en 20MHz-klocka och 16 KB RAM-minne [2]. En AVR processor kan utföra max en instruktion per klockcykel, vilket ger den en maximal beräkningskraft på 20 MIPS. På grund av detta måste man ha algoritmer som är ganska beräkningssnåla för att hinna utföra alla beräkningar på en rimlig tid. Avancerade karteringsalgoritmer har därför inte tagits upp i denna förstudie.

1.4 Metod

Denna förstudie kommer att utföras som en litteraturstudie. Information kommer hämtas från vetenskapliga publikationer och tidigare erfarenheter. Ett antal olika problem relaterade till styrmodulen beskrivs och möjliga lösningar presenteras. Välkända lösningsmetoder jämförs och relateras till det givna problemet.

1.5 Definitioner

Nödställd Tänkt nödställd i en labyrint som behöver undsättning. Kommer i verkligheten endast vara en speciellt markerad plats i labyrinten. Ibland refererat till som *målet*, *målruta* eller liknande.

Labyrint Den omgivning roboten ska utforska. Ibland kallad *omgivningen*, *banan* eller liknande.

Kartruta En 40x40 cm ruta. Labyrinten är uppbyggd av kartrutor.

Tillståndshierarki Struktur av logiska tillstånd som behövs för att utföra uppdraget. Tillstånden är uppdelade i tre nivåer.

Läge Högsta tillståndsnivån. Roboten kan vara i autonomt eller manuellt läge

Uppgift Mellersta tillståndsnivån. Ett övergripande tillstånd i den hierarkiska tillståndsmaskinen

Tillstånd Den lägsta tillståndsnivån. Utgörs av de mest elementära tillstånden i hierarkin.

Rutt En förutbestämd väg som roboten ska följa.

Micromouse En populär robottävlingsgren där man tävlar i att navigera i labyrinter. Då den delar flera principer med vårt uppdrag har en hel del inspiration hämtats därifrån.

2 Tillstånd

Robotens autonoma beslutsfattande uppstår ur fördefinierade tillstånd och tillståndsövergångar vilka förklaras nedan med en tillståndsmaskinsbeskrivning.

2.1 Tillståndsmaskinen

En tillståndsmaskin är en abstrakt modell för att beskriva ett beteende utifrån olika tillstånd och villkor för tillståndsövergångar. Modellen lämpar sig väl för att beskriva ett beslutsfattande system som beror på både aktuell information och historiska beslut, så som en autonom robot. En fördel med att konstruera beslutsfattandet som en tillståndsmaskin är att det ger en överskådlig kod där varje tillstånd enbart behöver beakta den delmängd beslutsdata som är relevant för just det tillståndet. En annan fördel är att det blir enkelt med testning och verifiering då man kan testa att varje enskilt tillstånd fungerar som det ska och sedan att tillståndsövergångarna sker på ett korrekt sätt.

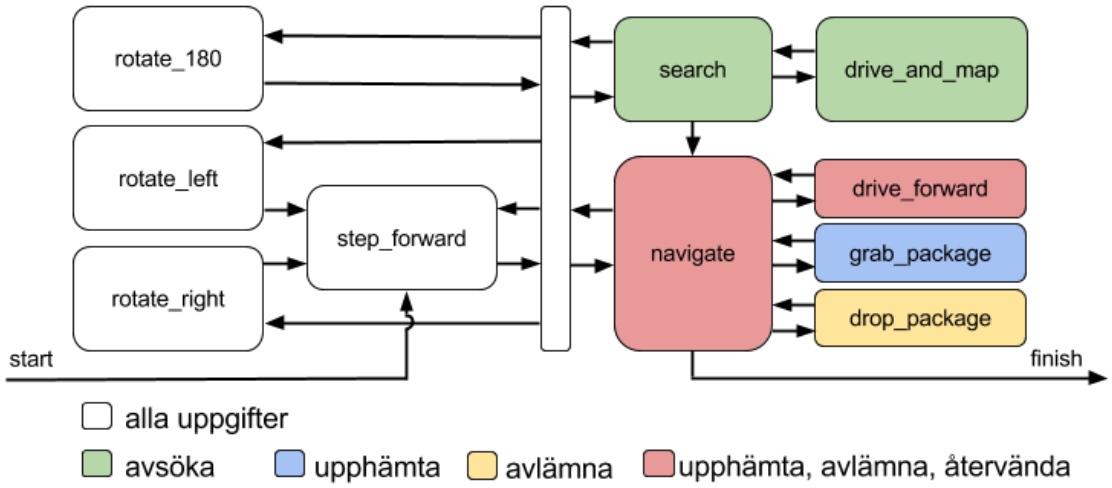
En avvägning som behöver göras vid realiseringen av en tillståndsmaskin är vad som ska utgöra ett eget tillstånd. Med för många tillstånd blir systemet svåröverskådligt och med för få, komplexa, tillstånd går tillståndsmodellens fördelar förlorade.

2.2 Tillstånd och tillståndshierarki

Ett relativt fullständigt tillståndsdiagram presenteras i bilaga 1. Ur figurerna framgår att det blir väldigt många tillstånd och flera av dem är dessutom väldigt lika. Givet uppgiftens utformning ligger det nära till hands att införa en hierarki [3] av tillstånd där robotens generella uppgift kan utgöra ett mer övergripande tillstånd, hädanefter kallat uppgift. Delstegen för att utföra uppgiften kan utgöra underordnade tillstånd. Som uppgifter kan man lämpligen välja avsöka, upphämta, avlämna och återvända. Dessa har ringats in i figurerna i bilaga 1. Tanken med uppgifterna är att låta tillståndens beteende bero av dessa och därmed kan antalet tillstånd och kodredundansen hållas låg.

Ett förslag till kompaktering presenteras i figur 1. Vissa tillstånd kommer att vara unika för en uppgift medan andra delas mellan flera. Uppgiften kan alltså diktera de andra tillståndens egenskaper och kriterier för tillståndsövergångar.

Dessa illustrationer visar bara hur det autonoma styrläget skulle fungera då det manuella är relativt simpelt.



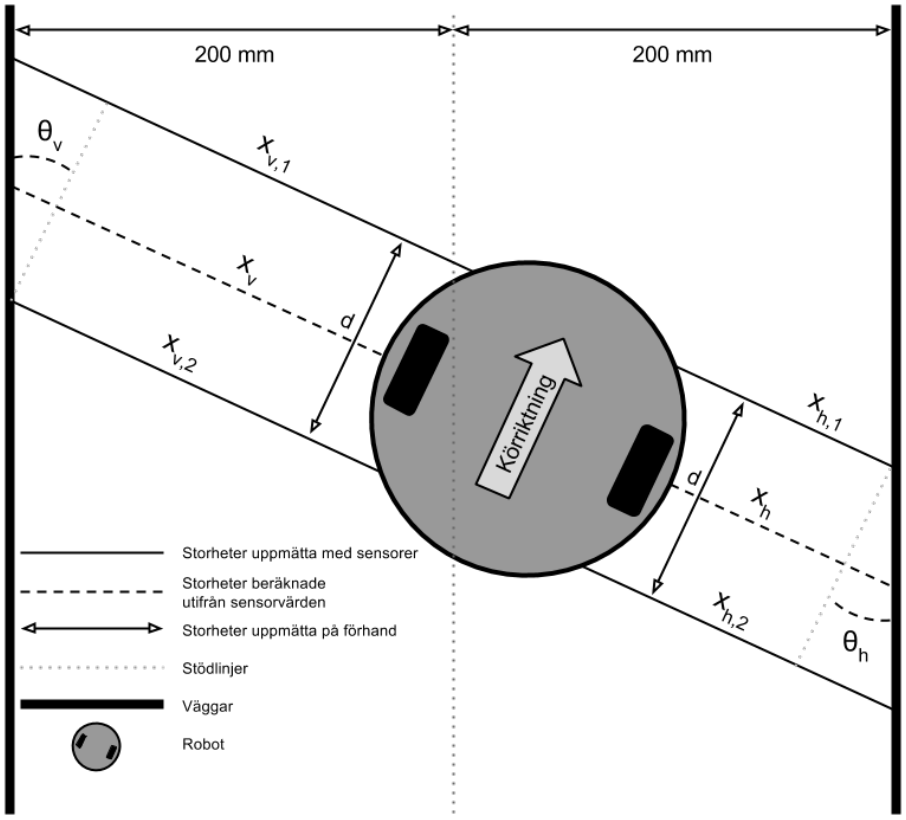
Figur 1: Fullständigt tillståndsschema för det autonoma läget. Varje ruta är ett tillstånd och färgerna anger vilka uppgifter som kan modifiera dessa tillstånd.

2.3 Realisering av en tillståndsmaskin som ett C-program

Det finns många sätt att realisera en tillståndsmaskin och på en ATmega1284P sker det lämpligen i form av ett C-program. Ett väldigt enkelt sätt att göra detta på är att låta varje tillstånd utgöra en egen C-funktion som returnerar en resultatkod till main-funktionen. Resultatkoden mappas mot en funktionspekare till ett annat tillstånd och på detta vis sker en tillståndsövergång. Uppgiften kan då anges av en global variabel som varje tillståndsfunktion kan ta hänsyn till eller ändra. Om det anses vara för långsamt att mappa resultatkod mot tillståndsfunktioner med vår begränsade beräkningskraft skulle man kunna undvika det genom den något mindre överskådliga varianten att funktionerna returnerar en funktionspekare till nästa tillstånd.

3 Reglering

Robotens finpositionering i korridorens mitt kan lämpligen uppnås genom återkoppling av sensordata. Ett enklare reglersystem av detta slag kan implementeras på många olika sätt och kan baseras på fysikaliska modeller eller uppmätta beteenden. Regleringen som är aktuell för denna robot ska i praktiken korrigera för skillnader mellan motorerna, ojämnheter i golv och andra små störningar som är svåra att modellera på ett bra sätt. Därför är det rimligt att experimentellt ta fram ett reglerystem som fungerar tillräckligt bra för de givna användarfallen. I figur 2 illustreras de sensorvärden som finns tillgängliga för styrmodulen och därmed kan användas av reglersystemet.



Figur 2: Roboten i en korridor med avstånd uppmätta av sensorerna utmarkerade. θ_v och θ_h samt x_v och x_h är beräknade värden utifrån $x_{v,1}$, $x_{v,2}$, $x_{h,1}$, $x_{h,2}$. d är ett på förhand uppmätt värde.

3.1 PD-reglering

Ett av de enklare reglersystemen som kan vara aktuellt är av PD-typ. Detta innebär att systemets styrsignal utgörs av en del som är proportionell mot reglerfelet, och en del som är proportionell mot derivatan av reglerfelet. Lämpligen är felet då robotens ortogonala avstånd till korridorens mitt och kan fås som differensen av avståndet till respektive sida. För att få fram derivata-termen behöver minst ett tidigare mätvärde

sparas för att göra en linjärapproximation av derivatan. Styrsignalen är lämpligen en term som adderas till ena motorns pulsbredd och subtraheras från den andras för att uppnå rotation. Denna term behöver begränsas på så sätt att det inte blir over- eller underflow på det 8-bitars tal som representerar pulsbredden. Detta är säkerligen tillräckligt för att hålla roboten i korridorens mitt, men man utnyttjar då inte att vi även har tillgång till en absolut vinkelmätning mot respektive vägg. Denna vinkel är information som systemet kan ha nytta av och frågan blir då hur man utnyttjar den.

Ett alternativ skulle vara att skapa två PD-regulatorer vars styrsignaler adderas tillsammans med olika viktning. Detta har gjorts med framgång i liknande system [4] men det tillför en del komplexitet till systemet och gör det något svårare att förutse.

En översikt över den aktuella sensorkonfigurationen kan ses i figur 2. Utifrån denna illustration framgår det att vinklar kan räknas ut enligt ekvation (1).

$$\theta_v = \arctan \frac{x_{v,1} - x_{v,2}}{d} \tag{1}$$

Eftersom trigonometriska uttryck kan vara tidskrävande att beräkna skulle man kunna använda småvinkelapproximation och reglera mot $\frac{x_{v,1}-x_{v,2}}{d}$ istället där d är en konstant som kan bakas in i reglerparametern. För att få ett värde på avståndet till mitten kan man för enkelhetens skull använda differensen $x_h - x_v$ vilken kommer att vara 0 när roboten är i mitten oavsett rotation. Däremot kommer differensen skilja sig från det ortogonala avståndet om roboten är roterad och förskjuten från mitten. Detta vinkelberoende torde inte ha allt för stor inverkan då roboten förhoppningsvis aldrig hinner hamna särskilt fel. För båda dessa mätvärden är referenssignalen 0. Vi har alltså en direkt mätning av vårt reglerfel.

Att använda PD-reglering på båda dessa reglerfel är något onödigt. Speciellt med den föreslagna småvinkelapproximationen eftersom differensen $x_{v,1} - x_{v,2}$ är starkt korrelerat med derivatan av det ortogonala avståndet. Så i någon mening är den uppskattade vinkeltermen en predikterande parameter för avståndet till mitten. Av den anledningen skulle man kunna använda proportionell reglering av de två ovan nämnda reglerfelen och se det som en typ av PD-reglering för avståndet till mitten.

3.2 Realisering av reglersystemet

Reglerparametrar kan tas fram experimentellt och konstanter i uttrycken kan bakas in i dessa. Sammanfattningsvis skulle ett enkelt reglersystem kunna implementeras enligt följande.

$$\begin{cases} v: \ hastighet\ (pulsbredd,\ motorer) \\ u: \ styrsignal \\ u_{max}: \ maximal\ styrsignal \\ e_x: \ reglerfel,\ position \\ e_\theta: \ reglerfel,\ vinkel \\ K_P,\ K_D: \ reglerparametrar \end{cases}$$

$$\begin{cases} u_{max} = \frac{v}{2} \\ e_x = (x_{v,1} + x_{v,2}) - (x_{h,1} + x_{h,2}) \\ e_\theta = (x_{v,1} - x_{v,2}) + (x_{h,1} - x_{h,2}) \\ u = \begin{cases} u_{max}, \ om \ K_P e_x + K_D e_\theta > u_{max} \\ -u_{max}, \ om \ K_P e_x + K_D e_\theta < -u_{max} \\ K_P e_x + K_D e_\theta, \ annars. \end{cases} \end{cases}$$

Detta är relativt enkelt att konstruera i styrmodulens mjukvara.

3.3 Vinkelderivata som övergångssignal

För att detektera en korsning i korridorssystemet kan man lämpligen övervaka derivatan av vinkelmätningar som görs mot respektive vägg, eller i praktiken, derivatan av differenserna $\delta x_v = x_{v,1} - x_{v,2}$ och $\delta x_h = x_{h,1} - x_{h,2}$. För att göra detta räknas förändringen gentemot föregående värde ut enligt ekvation (2) och (3) där T_s är den samplingstid styrmodulen samplar mätvärden från sensormodulen.

$$\Delta x_v = \left| \frac{\delta x_v - \delta x_{v,föregående}}{T_s} \right| \tag{2}$$

$$\Delta x_h = \left| \frac{\delta x_h - \delta x_{h,föregående}}{T_s} \right| \tag{3}$$

Δx_h och Δx_v kan då kontinuerligt jämföras mot ett experimentellt framtaget tröskelvärde för att detektera korridorskorsningar på respektive sida. Om tröskelvärdet överskrids så triggas en tillståndsövergång från drive_and_map eller drive_forward till tillståndet step_forward (alternativt i motsatt riktning). I praktiken innebär detta att reglersystemet för finpositionering i korridor slås av i korsningar. Detta kan göras under förutsättningen att robotens hastighet uppfyller (4) och därmed inte hinner få en sample med båda sensorerna mot väggen följt av en sample med båda sensorerna i den korsande korridoren.

$$v < \frac{d}{T_s} \tag{4}$$

4 Kartering och kortaste vägen

4.1 Introduktion

Kartering och kortaste vägen-problem är välkända och relativt väl studerade problem. Det finns ett antal olika ambitionsnivåer för kartering, till exempel om man vill kunna kartera godtycklig omgivning eller enbart specialfall. Det generella fallet kan vara mycket svårt och kräva mycket beräkningskraft. Eftersom omgivningen vi ska kartlägga följer väldigt specifika regler [5] kan en algoritm göras betydligt enklare.

En del inspiration har hämtats från robottävlingsgrenen micromouse. En micromouserobot är en liten robot som ska utforska en labyrint och hitta snabbaste vägen till mitten av labyrinten. Den har både likheter och olikheter med vårt uppdrag. Banan är uppbyggd av rutor och är vid starttillfället okänd i båda fallen, men för en micromouse är väggarna placerade mellan rutor istället för att hela rutorna är blockarade. Att målrutans position är känd i förväg för en micromouse skiljer sig också från vårt uppdrag. Vi kommer dock att titta mycket på micromousegrenen när vi tar fram våra karterings- och navigeringsalgoritmer.

4.2 Kartrepresentation

Kartan kommer i programkoden att representeras av en matris av bytes. I denna matris kommer varje element att motsvara en ruta i labyrinten. Olika värden på dessa element representerar olika saker i kartan. En möjlig representationsmodell är:

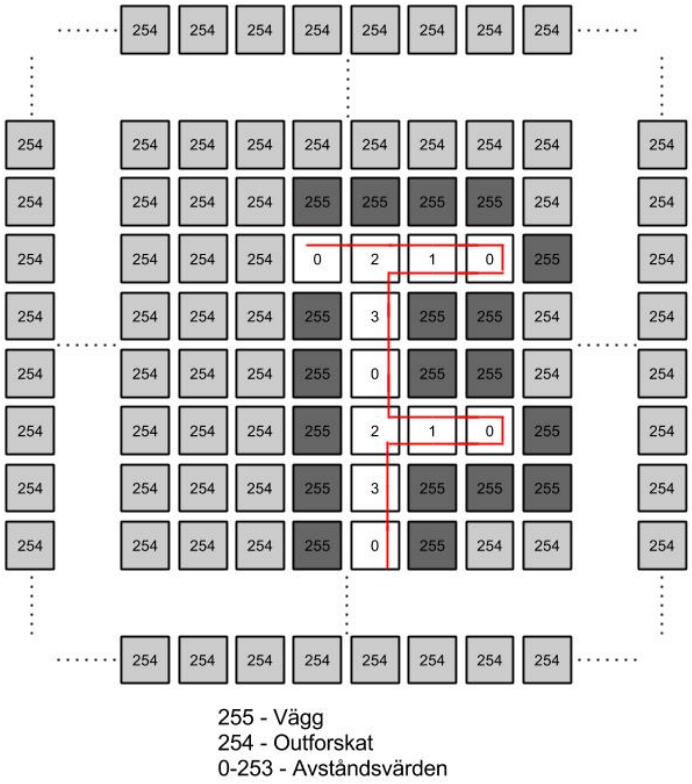
255 Icke farbar ruta. Sätts om man med sensorerna upptäcker en vägg.

254 Icke utforskad ruta, således också initialvärdet för alla rutor.

0-253 Värden som sätts i och med kortaste vägen-beräkningar på farbara rutor. Är endast giltiga i samband med att en kortaste vägen-beräkning har utförts.

Då labyrinten enligt banspecifikationen [5] har en maximal storlek kan man i förhand göra matrisen lagom stor för att hela labyrinten ska kunna karteras. Detta är smidigt då en datastruktur som dynamiskt växer under karteringen är besvärligare att implementera. Minnesanvändningen för denna matris är inte heller obekvämt stor, en matris på 32x32 element á en byte är mer än tillräckligt och tar då upp en kilobyte minne. Troligen kommer vi klara oss på en matris av ungefär halva storleken.

En illustration av kartrepresentationen ges i figur 3



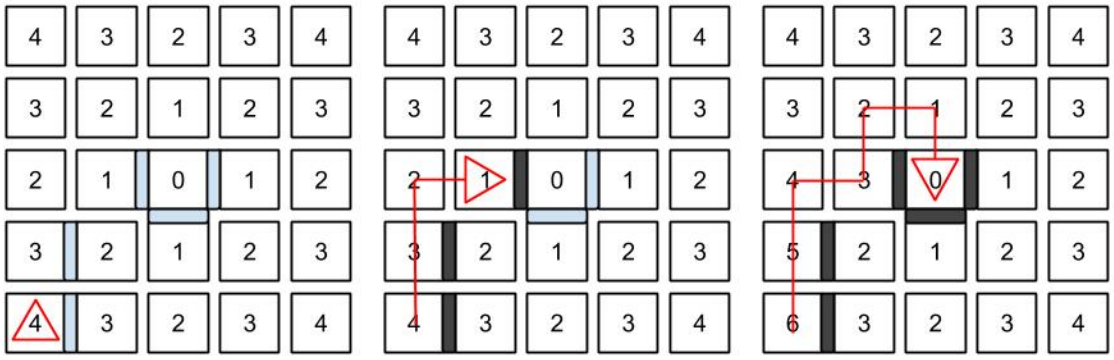
Figur 3: Illustration av kartrepresentationen, med rutt för roboten inritad.

4.3 Väggföljande algoritm

En mycket enkel algoritm för att ta sig igenom en labyrint är s.k. väggföljande algoritmer (“right/left hand follower” eller “right/left hand on the wall”) [6]. I denna algoritm följer man hela tiden antingen höger eller vänster vägg, i likhet med att gå framåt och släpa handen mot väggen. Denna algoritm är garanterad att lösa vissa typer av labyrinter, nämligen de som är enkelt sammanhängande. Om labyrintens väggar inte är enkelt sammanhängande, till exempel om det finns öar och eller loopar, är denna metod inte garanterad att fungera. Då omgivningen i vårt fall inte är garanterad att vara enkelt sammanhängande är inte en väggföljande algoritm ett alternativ.

4.4 Flood-fill

En mycket vanlig algoritm för labyrintutforskning för Micromouse är flood-fill [6]. Denna algoritm finns i olika variationer men gemensamt för dem alla är att varje ruta tilldelas ett nummer baserat på hur långt man måste färdas från en viss ruta för att komma till denna ruta. Roboten måste också lagra vilka rutor man kan köra i, eller vilka väggar som finns i labyrinten. När man börjar avsökningen börjar man med en karta helt utan väggar och lägger till dem vartefter de upptäcks.



Figur 4: Exempel på flood-fill för en micromouse.

För en micromouse-robot vet man att målet finns i de fyra rutorna i mitten och att man börjar i ett hörn. Man kan då börja med att göra en optimistisk flood-fill, alltså anta att labyrinten inte innehåller några väggar och fylla var ruta med avståndsvärden. Rutorna i mitten får värdet 0, alla rutor som gränsar till dessa (dock inte diagonalt) får värdet ett och så vidare. På så vis får man en optimistisk skattning av hur långt man måste köra för att komma in till målet från alla rutor i labyrinten. Algoritmen blir i detta fall att hela tiden flytta sig till den angränsade ruta som har lägst avståndsnummer. Finns det flera rutor får man välja en enligt någon metod, t.ex favorisera att fortsätta rakt fram. Efter varje flytt uppdaterar man väggarna omkring rutan och kontrollerar om numreringen fortfarande är giltig. Upptäcker man en vägg mellan den rutan man står i och den som har avståndsnummer en enhet lägre kan man dra slutsatsen att ens tidigare flood-fill inte är sann. Man får då göra om processen och återigen flytta sig till ett lägre ordningsnummer tills dess att man funnit målet. En illustration av processen visas i figur 4.

4.4.1 Applicerbarhet på vårt uppdrag

Ett par skillnader i vårt uppdrag kontra micromouse-fallet gör att denna algoritm inte är applicerbar rakt av. Vi kommer inte veta var i labyrinten vi börjar och vi vet inte heller var målet är. Detta gör att vi inte kan göra dessa optimistiska skattningar om hur lång det är till målrutan. Vi har då inte en given punkt att sträva mot utan måste utforska labyrinten till dess att målrutan är funnen. En fördel vore om man kunde hålla nere antalet överflödiga beräkningar för att spara på mikrokontrollerns beräkningskapacitet.

De algoritmer som ofta beskrivs är anpassade för en micromousebana. I dessa banor är väggarna placerade mellan rutorna och de allra flesta rutorna går att nå. Detta ger upphov till en datastruktur för varje ruta som innehåller 5 värden, avståndsvärdet samt vilka av rutans sidor som har väggar[7]. I vår bana kommer hela rutor antingen att vara körbara eller inte. Man behöver alltså inte en datastruktur som kan lagra så många olika värden för var ruta som i fallet för en micromouserobot.

Att använda flood-fill för kortaste vägen-beräkningar är däremot väl applicerbart på vår situation. Då en flood-fill-algoritm ser lite olika beroende på implementation kan man inte säga att vi kan använda algoritmen rakt av. Principen för flood-fill är

dock användbar även om anpassningar till vår implementation måste göras. I micromousefallet utforskar man labyrinten genom optimistiska gissningar av närmaste vägen till målet och följa denna så länge som möjligt. Hittar man väggar gör man en ny skattning och kör efter den nya rutten. Vi kan inte använda denna metod då målets position inte är känd i förväg. Istället kan vi hela tiden röra oss till en angränsande outforskad ruta och samtidigt övervaka vår omgivning.

När vi kommer i situationen att ingen outforskad ruta finns som granne vill vi göra en flood-fill och ta fram vägen till närmaste outforskade ruta. Algoritmen fyller på med värden till dess att ”vågfronten”kommit fram till en outforskad ruta. När detta sker är vi inte intresserade av att fortsätta algoritmen och den kan avbrytas. Detta gör att rutor som inte behandlades av senaste kortaste vägen-beräkningen kommer ha ogiltiga värden och det är därför viktigt att inte använda gamla värden ur kartrepresentationen för navigation.

5 Resultat

5.1 Tillstånd och reglering

Att konstruera ett autonomt beslutsfattande för en robot kan vid första anblick verka svårt. Men förutsatt att roboten endast kan ställas inför ett begränsat antal fördefinierade situationer blir det enkelt att strukturera upp beslutsfattandet till exempel med någon form av tillståndsmaskin. För att hålla antalen tillstånd lågt kan man dela upp dem i hierarkier. Det är enkelt att realisera en tillståndsmaskin med C-kod. Tillståndsmaskinen som presenteras i denna förstudie kommer säkerligen behöva anpassas till verkligheten när den skall realiseras.

Det finns många sätt att konstruera ett reglersystem på och i detta fall borde ett som är skapat utifrån de sensorer och beräkningsmässiga begränsningar vi har fungera tillräckligt bra. En PD-reglering som inte skattar reglerfelets derivata numeriskt utan låter ett separat mätvärde vara den prediktiva termen i regleringen föreslås. Eftersom mätsignaler har valts utifrån hur praktiska de är att beräkna så blir det inte helt uppenbart hur olika reglerparametrar kommer att påverka systemet. Därför kommer alla reglerparametrar att tas fram experimentellt och förhoppningsvis fungerar systemet just *tillräckligt* bra.

5.2 Kartering och kortaste vägen

Vi har kommit fram till att för en micromouse används nästan uteslutande någon form av flood fill-algoritm för navigering i labyrinten. Implementeringen varierar mellan olika robotar och det presenteras många alternativ för hur man kan försöka optimera eller få ut så mycket information som möjligt. Vi har kommit fram till att vår karta kan representeras med en enkel matris som lagrar både data om kartan och data för kortaste vägen-beräkningar. Denna representation är möjlig då banan har restriktioner på sitt utseende [5] som gör den enkel att representera på ett kompakt vis. Att representera en generell miljö är ett svårare problem.

För att fylla på information i kartan konstruerar man en algoritm som enligt ett mönster avsöker alla rutor och med sensorerna kollar på omgivningen. Att så länge som möjligt röra sig till den närmaste oavsökta rutan kommer göra att man till slut har avsökt hela labyrinten. Oavsökta rutor som är granne till den ruta roboten står i är mycket enkla att upptäcka och ta sig till. Om en sådan ruta inte existerar måste man finna den närmaste oavsökta rutan och beräkna kortaste vägen dit. För detta är flood-fill en lämplig algoritm som kan köras tills oavsökt ruta är funnen. När detta har skett är det enkelt att få ut den kortaste rutten till denna ruta utifrån de beräkningar som gjorts av algoritmen genom att rekursivt stega sig tillbaka till rutor med lägre avståndsnummer till rutan roboten står i.

Referenser

[1] Grupp1-6, ”Tävlingsregler”, Version 1.0, febr. 2015.

[2] *Atmega1284p datasheet*, ATmega1284p, Rev. 8059D, Atmel, nov. 2009.

[3] N. P. Dash, R. Dasgupta, J. Chekapa och A. Halder, ”Event driver programming for embedded systems - a finite state machine based approach”, i *Innovation Lab, Tata Consultancy Services Ltd. Kolkata*, 2011.

[4] S. G. Kibler, A. E. Hauer, D. S. Giessel, C. S. Malveaux och D. Raskovic, ”IEEE micromouse for mechatronics research and education”, i *2011 IEEE International Conference on Mechatronics*.

[5] Grupp1-6, ”Banspecifikation”, Version 1.0, febr. 2015.

[6] S. Mishra och P. Bande, ”Maze solving algorithm for micro mouse”, i *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*, 2008.

[7] L. Wyard-Scott och Q.-H. M. Meng, ”A potential maze solving algorithm for a micromouse robot”, i *1995 IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*.

A Bilaga 1