LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Musen

Digitala Projekt SK 2005

Författare:
Joel Guedj e02jg
Erik Dahlbäck e02ed

Handledare:
Bertil Lindvall

Innehåll

Inledning

Mål

Uppgiften går ut på att konstruera en mus (LEGO), som hittar kortaste vägen från start till mål i en godtycklig labyrint.
Några hållpunkter:

- Labyrinten består av moduler med måtten 20 X 20 cm. Hela labyrinten täcker maximalt 4 X 4 m.
- Musen kan använda sig av optiska, akustiska eller mekaniska sensorer.
- All styrlogik och kraftförsörjning skall finnas på kortet.

Under körningen får inte batteribyte ske. Musen skall klara sig på en laddning från startskott till målgång. Genomsökning av labyrinten och körningen får tar max 15 min.

Överblick

Musen i sig är helt mekanisk men den styrs av en dator, en AVR. AVR:en tar emot data från givarna och behandlar den. Den skickar även ut data som styr motorer, lysdioder och så vidare.

I bilaga 1 finns bilder på musen för att underlätta läsningen. Källkoden (hittills) redovisas i bilaga 2.

Planering

Vi började med att brainstorma lite om hur musen skulle byggas rent fysiskt och vilken sorts styrning den skulle ha. Programmeringsbiten väntade vi med för stunden.

Styrning

Vi ville ha en styrning som medförde att musen smidigt och enkelt kunde svänga runt i labyrinten utan att fastna i väggar och hörn. Så liten svängradie som möjligt var även önskvärt.

Kaross

För att underlätta manövrering i labyrinten gällde det att få musen så liten och kompakt som möjligt. Hellre bygga på höjden än på bredden och lägga så mycket tyngd i botten som möjligt för att få den stabil.

Positionering

Med själva musen planerad, var det dags att komma på hur vi skulle kunna positionera den i labyrinten. Alltså helt enkelt hålla reda på var musen står och åt vilket håll den pekar. En annan sak som behövdes var någon sorts mekanism som höll musen borta från att köra snett och åka in i väggarna.

Slutligen kom vi fram till att vad som behövdes var:

- Avståndsmätare som mätte avståndet till väggarna på sidorna och framåt.
- Trippmätare som mäter avverkad sträcka.
- Vinkelgivare som håller reda på vart musen pekar.

Programmering

Labyrinten representeras som en 3-dimensionell matris, där varje ruta och dess tillhörande väggar kan beskrivas. Genom att implementera Dijkstras algoritm kan kortaste väg mellan två noder (korsningar) beräknas.

En sorts regulator för att hålla musen i mitten på vägen planerades också att skrivas.

Utförande

Styrning

Flera olika prototyper byggdes, och deras svängförmåga och svängradie testades i labyrinten. Bandvagnsprincipen, vanlig personbilsstyrning med differential-axel och en modell med styrning på både fram- och bakhjulen testades. Vad vi till slut valde var

bandvagnsprincipen, men med hjul istället för larvfötter. Två motorer styr varsitt hjulpar på höger och vänster sida. Genom att köra ena hjulparet framåt och det andra bakåt, kan man få musen att svänga på stället.

Kaross

Eftersom vi inte bara ville ha en kompakt konstruktion utan även en hög utväxling, för att kunna få en stark drivning, krävdes det många kuggar och axlar. Dessa tar upp mycket plats och vi fick verkligen nytta av våra legokunskaper från yngre år och vi fick till slut en hög, kompakt mus med låg tyngdpunkt.

Positionering

För avståndsbedömningen valde vi att använda oss av IR- och fotodioder. IR-dioderna emmiterar strålning som snappas upp av fotodioderna. Ju närmre en vägg är, desto mer reflekterar den och vi får då en starkare signal. Eftersom vi har tre givare som ska kollas i princip samtidigt, uppstår det problem. Givarna är kopplade till AVR:ens AD-omvandlare, denna behöver en viss tid för att ställa in sig innan den kan omvandla, vilket vi inte tog hänsyn till i början och fick därför många fel. Detta upptäcktes dock och en delay lades in mellan de tre olika mätningarna. Problem med oönskad reflektion uppstår fortfarande men, det är överkomligt.

I musen finns en trippmätare som består av en halv roterande skiva, en röd lysdiod och en fotodiod. Fotodioden detekterar ljuset som släpps igenom varje varv och signalen AD-omvandlas. I AVR:en räknar vi ut hur många varv skivan snurrar och vet därför hur långt vi har åkt, eftersom skivan är direkt kopplad till drivningen.

Vi kom över en digital kompass som fungerade utmärkt som vinkelgivare. Den var mycket noggrann och programvaran vi skrev till den fungerade väl. Till vår besvikelse uppstod problem när vi kopplade den till musen, den verkade störas av något. Vi antar att detta berodde på batteriets magnetiska förmåga. Eftersom vi verkligen behövde något som angav hur vi var vridna (för att veta om vi svängt 90° tex.), fick vi tänka om. Vi kom på att hjulen snurrar när vi svänger och att vår trippmätare snurrar med dem. Genom att ta reda på hur många ”tics” som motsvarade t.ex. 90°, kunde vi trots allt svänga med ganska god precision genom att titta på trippmätaren.

Programmering

Som första steg gällde det att lära känna AVR:en och flera små testfunktioner skrevs för att vi bättre skulle förstå handhavandet. När principerna för skrivning och läsning av portar osv. hade sjunkit in och

vi testat dem på styrningen av motorerna, började testerna på resten av musen.

Nästa del av musen som skulle programmeras var avståndsbedömningen. När musen startas ska det alltid köras en initiering, som bestämmer värden för öppen väg, vägg eller nära. Detta eftersom ljuset i rummet kan variera och därför kan olika värden mätas från gång till gång. Så initieringen sätter vilka gränser som gäller för de tre olika avstånds fallen och dessa gäller sen för resten av körningen. Till vår hjälp konstruerade vi ett set med lampor som lyser och indikerar vad varje sida ”ser”.

När väl avståndsbedömningen fungerade som den skulle, var det dags att ta sig an problemet med att köra, läsa av och minnas. Labyrinten representeras av en 3-dimensionell matris, där xy-planet definierar olika rutor och i z-led definieras olika attribut för just en ruta. Varje ruta har bland annat attribut för de fyra olika väggarnas existens (2 = ingen vägg, 1 eller 0 = vägg, ∞ = vet inte). Musen fungerar på så sätt att den kör en ruta framåt, läser av de tre väggarna den kan se, uppdaterar rutan vi står i och uppdaterar även omringande rutor. Detta är själva principen för hur ett steg i avsökningen (eller vanlig körning) fungerar.

Avståndsregulatorn som skulle implementeras visade sig lite för krånglig, så vi började leta efter en mekanisk lösning istället. Fram på musen sitter nu två armar med hjul på som ser till att musen rätas ut vid eventuell krock mot vägg. De fungerar utmärkt.

Alla steg fungerar var för sig, men nu gäller det att använda dem tillsammans för att implementera Dijkstras algoritm. I skrivande stund är inte det gjort än, men beräknas vara klart till den muntliga redovisningen. Dock har mindre testprogram skrivits där musen läser av delar av labyrinten och kan sedan minnas vad den sett (tex ”sväng andra till höger”, utan att använda avståndsmätarna).

Komplikationer med hårdvaran

På grund av dålig precision i hårdvaran fungerar inte alla delarna av musen så bra i praktiken som de gör i teorin. Ger inte hårdvaran värden som håller sig konstant inom vissa gränser, så är det svårt att kompensera detta med mjukvaran. Därför beter sig musen inte alltid på samma sätt.

Avståndsmätarna beror (trots initieringen) på vilket ljus det är i rummet och kan lätt missta en vägg för en öppning. De störs även av reflektion, efter både sändare och mottagare jobbar i ett relativt brett synfält. Tester med skärmning utfördes, men gav inget positivt resultat. Detta leder till att musen inte alltid läser av labyrinten rätt.

Trippmätarens upplösning leder också till problem. Ett varv motsvarar ca 2 centimeters körning, och vi får därför en felmarginal på 1 cm. Kör

man en längre sträcka blir det lätt att man ”kommer ur fas”. Problemet hade kunnat lösas genom att köra någon sorts check gentemot öppningar och hörn osv. I vårt fall kommer vi rätt varje gång vi ändrar riktning. Samma problem gäller när vi svänger.

Trippmätarens upplösning hade kunnat förbättras genom att låta den snurra snabbare, men eftersom vi AD-omvandlar utsignalen är vi rädda för att missa varv pga. slöhet i omvandlingen om vi gör det. En fördel hade vart att slippa AD-omvandla utsignalen men vi kunde inte uppnå någon bra digital signal.

Dessa komplikationer leder tillsammans till att musen ibland är opålitlig och svänger för mycket, för lite, för tidigt, för sent, in i en vägg eller inte alls.

Koppling av hårdvaran

För att driva motorerna används en H-brygga eftersom AVR:en inte klarar av att leverera tillräckliga strömmar. H-bryggan matas separat och har ett par styrportar kopplade till AVR:en.

Fotodioderna är kopplade seriellt med ett motstånd och matas med 5V. Vid påverkan av IR-ljus ändrar dioden strömmen i kretsen och vi kan mäta upp en differens över motståndet. Signalen AD-omvandlas och tas om hand. Trippmätaren fungerar på samma sätt och drivs med en kedja från högra hjulparet.

IR-sändare drivs med transistorer kopplade till AVR:en (vi trodde det behövdes, men det verkar inte så trots allt). Strömmen i dioden begränsas med ett motstånd.

Lamporna i testdisplayen drivs också de med transistor och motstånd.

Resultat och diskussion

Sammanfattningsvis så fungerar musen bra i teorin men på grund av problem men hårdvaran och tidsbrist uppför den sig inte på önskvärt sätt, och därmed är inte målen uppfyllda.

Hade vi fått chansen att göra om projektet från början så är vi övertygade om att alla mål hade blivit uppfyllda eftersom felsökning och inlärning har tagit majoriteten av tiden. Vi har alltid vetat vad vi velat uppnå men det är den praktiska biten som har tagit tid.

Projektet har givit oss inblick i hur det är att arbeta med prototyper och utveckling. En bra idé är svår att realisera utan att stöta på många svåra problem. Resultatet blir ofta en kompromiss.

Bilaga 1

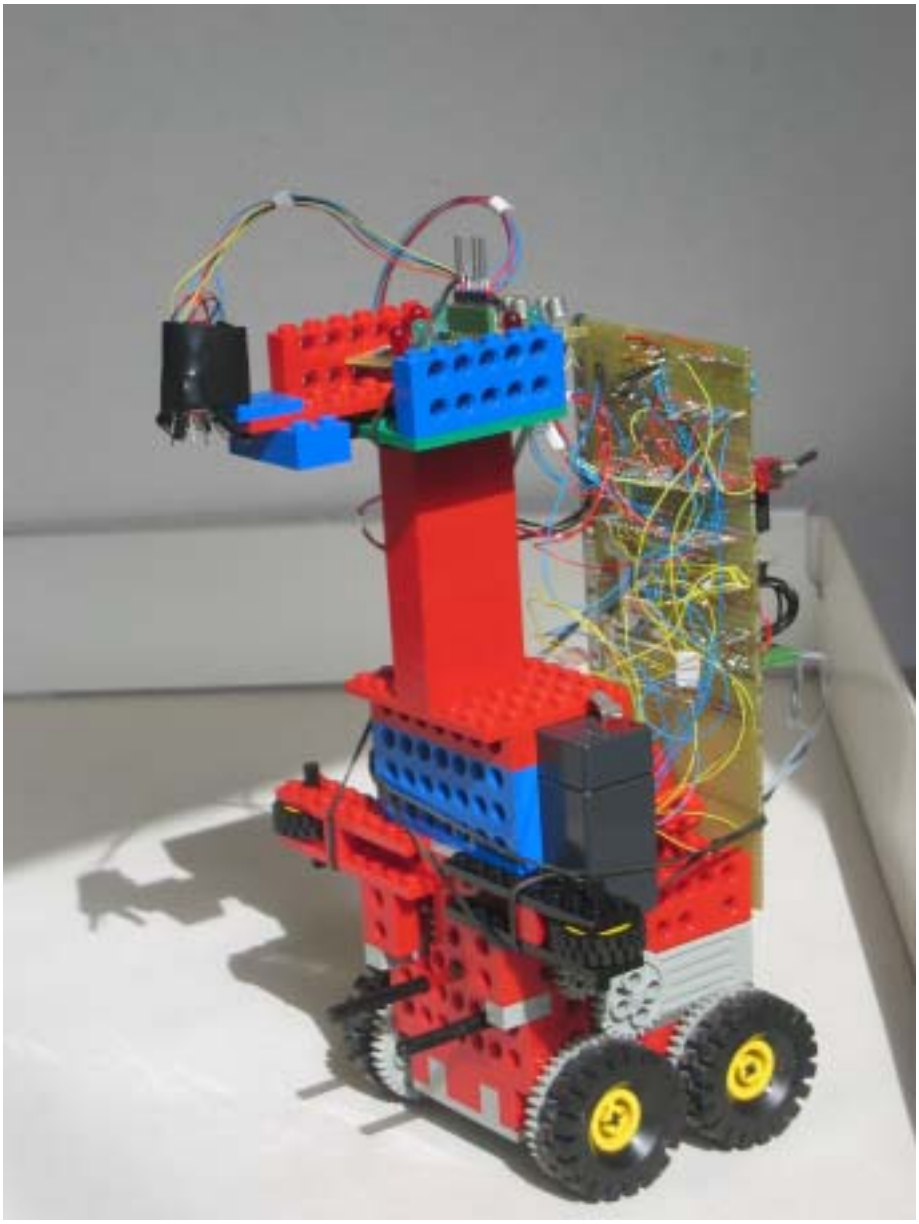

Bild 1

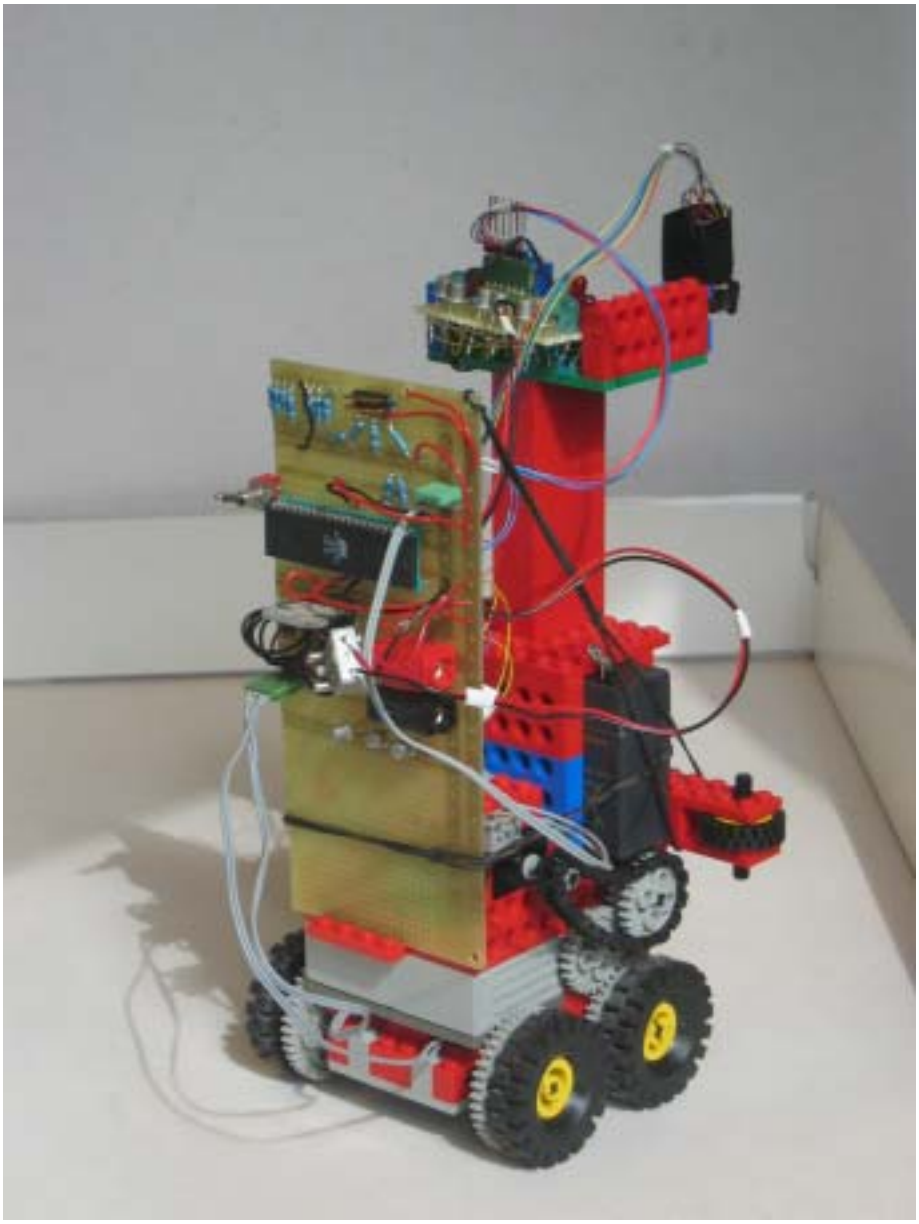

Bild 2

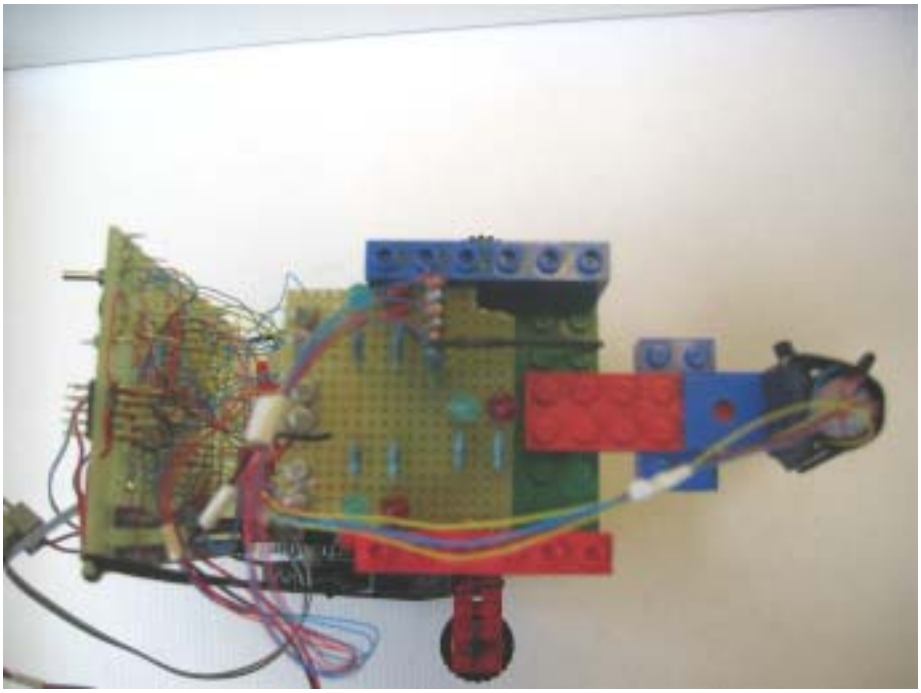

Bild 3

Bild 4

Bilaga 2

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

unsigned char c;
unsigned int low,high;

int standNorth = 0;
int standEast = 0;
int standSouth = 0;
int standWest = 0;
int tempA = 0;
int tempB = 0;
int close, far;
int nbr = 0;
int labyrint[8][11][4];       // labyrinten, x,y och med
fyra väggar
int standing = 0;             // säger vårt väderstreck:
0 = norr, 1 = öst, 2 = syd, 3 = väst
int x = 4;                    // startkoordinater
int y = 11;                   // startkoordinater
// labyrintens väggar
int northWall, eastWall, southWall, westWall;

// svänger vänster 91 grader
void turnLeft(){
  int temp;
  int last = 0;
  int tics = 7;

  while(tics > 0){  // program för att snurra x antal
varv och sen stänga av motorn
    ADMUX = 0x44;                          // ställ in ADn
till varvräknaren
    delay_ms(500);
    ADCSR = 0xC0;                          // initiera ADn
    PORTB = 0x3A;                          // kör bilen
framåt
    while((ADCSR && ADIF) != 1){}   //gör så att vi
väntar på att ADn är klar
    low = ADCL;
    high = ADCH;

    temp = high * 256 + low;  // lagra det nya värdet i
temp

    if(temp < 800){           // är värdet litet
      temp = 0;               // sätt till 0
      if(abs(temp - last) == 1){    // har vi bytt värde
sen senast ( = ett varv)
      tics = tics - 1;        // räkna ner tics
      last = temp;                  // ändra last
      }
    }else if(temp > 890){     // är värdet stort
      temp = 1;                          // sätt till 1
      if(abs(temp - last) == 1){    // samma som ovan
      tics = tics - 1;        // --"--
      last = temp;            // --"--
      }
    }
  }

  PORTB = 0;                  // stäng av motorn när vi
är klara
}
// svänger höger 91 grader
void turnRight(){
```

```
  int temp;
  int last = 0;
  int tics = 8;

  while(tics > 0){  // program för att snurra x antal
varv och sen stänga av motorn
    ADMUX = 0x44;       // ställ in ADn till varvräknaren
    delay_ms(500);
    ADCSR = 0xC0;       // initiera ADn
    PORTB = 0x35;       // kör bilen framåt
    while((ADCSR && ADIF) != 1){}   //gör så att vi
väntar på att ADn är klar
    low = ADCL;
    high = ADCH;

    temp = high * 256 + low;  // lagra det nya värdet i
temp

    if(temp < 800){           // är värdet litet
      temp = 0;                           // sätt till 0
      if(abs(temp - last) == 1){    // har vi bytt värde
sen senast ( = ett varv)
      tics = tics - 1;        // räkna ner tics
      last = temp;            // ändra last
      }
    }else if(temp > 890){     // är värdet stort
      temp = 1;               // sätt till 1
      if(abs(temp - last) == 1){    // samma som ovan
      tics = tics - 1;        // --"--
      last = temp;            // --"--
      }
    }
  }

  PORTB = 0;      // stäng av motorn när vi är klara
}

// kör framåt tics antal steg

void forward(int tics){
  int temp;
  int last = 0;

  while(tics > 0){  // program för att snurra x antal
varv och sen stänga av motorn
    ADMUX = 0x44;       // ställ in ADn till varvräknaren
    delay_ms(500);
    ADCSR = 0xC0;       // initiera ADn
    PORTB = 0x36;       // kör bilen framåt
    while((ADCSR && ADIF) != 1){}    // gör så att vi
väntar på att ADn är klar


    low = ADCL;
    high = ADCH;

    temp = high * 256 + low;  // lagra det nya värdet i
temp

    if(temp < 800){                 // är värdet litet
      temp = 0;                     // sätt till 0
      if(abs(temp - last) == 1){    // har vi bytt värde
sen senast ( = ett varv)
      tics = tics - 1;        // räkna ner tics
      last = temp;                  // ändra last
      }
    }else if(temp > 890){           // är värdet stort
      temp = 1;                     // sätt till 1
      if(abs(temp - last) == 1){    // samma som ovan
```

```
      tics = tics - 1;        // --"--
      last = temp;            // --"--
      }
    }
  }

  PORTB = 0;      // stäng av motorn när vi är klara
}
void back(int tics){
  int temp;
  int last = 0;

  while(tics > 0){  // program för att snurra x antal
varv och sen stänga av motorn
    ADMUX = 0x44;       // ställ in ADn till varvräknaren
    delay_ms(500);
    ADCSR = 0xC0;       // initiera ADn
    PORTB = 0x39;       // kör bilen framåt
    while((ADCSR && ADIF) != 1){  }     // gör så att vi
väntar på att ADn är klar

    low = ADCL;
    high = ADCH;

    temp = high * 256 + low;  // lagra det nya värdet i
temp

    if(temp < 800){           // är värdet litet
      temp = 0;               // sätt till 0
      if(abs(temp - last) == 1){    // har vi bytt värde
sen senast ( = ett varv)
      tics = tics - 1;        // räkna ner tics
      last = temp;            // ändra last
      }
    }else if(temp > 890){     // är värdet stort
      temp = 1;               // sätt till 1
      if(abs(temp - last) == 1){    // samma som ovan
      tics = tics - 1;        // --"--
      last = temp;            // --"--
      }
    }
  }

  PORTB = 0;      // stäng av motorn när vi är klara
}



// kolla avstånd framåt, returnerar distance
int checkFront(){
  int front;
  int distance;   // distance: 0 = nära, 1 = en bit bort,
2 = oändligheten
  int vect[4];

  sbi(PORTD, 1);  // sätt biten på plats 1 (starta den
främre sändaren), kan behöva delayas för att hinna tändas

  for(int i = 0; i < 4; i++){ // Ta två värden och få ett
medelvärde
    ADMUX = 0x41;             // ställ in ADn till främre
mottagaren
    delay_ms(500);
    ADCSR = 0xC0;             // initiera ADn

    while((ADCSR && ADIF) != 1){}   // gör så att vi
väntar på att ADn är klar, maska med ADIF

    low = ADCL;
```

```
    high = ADCH;
    vect[i] = high * 256 + low;
  }
  cbi(PORTD, 1);              // stäng av den främre
sändaren

  front = vect[0] + vect[1] + vect[2] + vect[3];
  //temp >> 2;                // shifta ett steg höger,
vilket motsvarar division med 2;
  front = front / 4;

  if(front > close){          // kolla vilket avstånd vi
har och sätt distance därefter
    distance = 0;             // nära
    cbi(PORTA, 7);            // släck lampor fram när vi
är nära
    cbi(PORTD, 7);            // släck lampor fram när vi
är nära
  }else if(front >= far && front <= close){
    distance = 1;             // en bit bort
    sbi(PORTA, 7);            // tänd grön fram för ok
avstånd
    cbi(PORTD, 7);            // släck röd fram för ok
avstånd
  }else if(front < far){
    distance = 2;             // oändligheten
    sbi(PORTD, 7);            // tänd röd fram för
oändligheten
    cbi(PORTA, 7);            // släck grön fram för
oändligheten
  }


  return distance;
}

// kolla avstånd vänster, returnerar distance
int checkLeft(){
  int left;
  int distance;         // distance: 0 = nära, 1 = en bit
bort, 2 = oändligheten
  int vect[4];

  sbi(PORTD, 0);        // sätt biten på plats 1 (starta
den främre sändaren), kan behöva delayas för att hinna
tändas

  for(int i = 0; i < 4; i++){ // Ta två värden och få ett
medelvärde
    ADMUX = 0x40;             // ställ in ADn till främre
mottagaren
    delay_ms(500);
    ADCSR = 0xC0;             // initiera ADn

    while((ADCSR && ADIF) != 1){}   // gör så att vi
väntar på att ADn är klar, maska med ADIF

    low = ADCL;
    high = ADCH;
    vect[i] = high * 256 + low;
  }
  cbi(PORTD, 0);              // stäng av den främre
sändaren
  left = vect[0] + vect[1] + vect[2] + vect[3];
  //temp >> 2;
  left = left / 4;

  if(left > close){           // kolla vilket avstånd vi
har och sätt distance därefter
```

```
    distance = 0;       // nära
    cbi(PORTA, 5);      // släck vänster lampor när vi är
nära
    cbi(PORTA, 6);      // släck vänster lampor när vi är
nära
  }else if(left >= far && left <= close){
    distance = 1;       // en bit bort
    sbi(PORTA, 5);      // tänd grön vänster för ok
avstånd
    cbi(PORTA, 6);      // släck röd vänster
  }else if(left < far){
    distance = 2;       // oändligheten
    sbi(PORTA, 6);      // tänd röd vänster för
oändligheten
    cbi(PORTA, 5);      // släck grön vänster
  }



  return distance;
}
// kolla avstånd höger, returnerar distance
int checkRight(){
  int right;
  int distance;         // distance: 0 = nära, 1 = en bit
bort, 2 = oändligheten
  int vect[4];

  sbi(PORTD, 2);        // sätt biten på plats 1 (starta
den främre sändaren), kan behöva delayas för att hinna
tändas



  for(int i = 0; i < 4; i++){ // Ta två värden och få ett
medelvärde
    ADMUX = 0x42;             // ställ in ADn till främre
mottagaren'
    delay_ms(500);
    ADCSR = 0xC0;             // initiera ADn

    while((ADCSR && ADIF) != 1){}   // gör så att vi
väntar på att ADn är klar, maska med ADIF

    low = ADCL;
    high = ADCH;
    vect[i] = high * 256 + low;
  }
  cbi(PORTD, 2);        // stäng av den främre sändaren
  right = vect[0] + vect[1] + vect[2] + vect[3];
  //temp >> 2;
  right = right / 4;

  if(right > close){    // kolla vilket avstånd vi har
och sätt distance därefter
    distance = 0;       // nära
    cbi(PORTC, 0);      // släck höger lampor när vi är
för nära
    cbi(PORTC, 1);      // släck höger lampor när vi är
för nära
  }else if(right >= far && right <= close){
    distance = 1;       // en bit bort
    sbi(PORTC, 0);      // tänd grön höger när vi är på
rätt avstånd
    cbi(PORTC, 1);      // släck röd höger
  }else if(right < far){
    distance = 2;       // oändligheten
    sbi(PORTC, 1);      // tänd röd höger för
oändligheten
```

```
    cbi(PORTC, 0);      // släck grön höger
  }


  return distance;
}

//initieringsprogram, vänster mot en vägg, höger mot
oändligheten
void init(){
  int vect[4];

  sbi(PORTD, 0);        // starta vänster sändare
  delay_ms(500);
  delay_ms(500);


  for(int i = 0; i < 4; i++){       // Ta två värden och
få ett medelvärde
    ADMUX = 0x40;                   // ställ in ADn till
vänster
    delay_ms(500);
    ADCSR = 0xC0;                   // initiera ADn

    while((ADCSR && ADIF) != 1){}   // gör så att vi
väntar på att ADn är klar, maska med ADIF

    low = ADCL;
    high = ADCH;
    vect[i] = high * 256 + low;
  }
  cbi(PORTD, 0);  // stäng av vänster sändare
  close = vect[0] + vect[1] + vect[2] + vect[3];
  //close >> 2;         // skifta två steg (dela med
fyra)
  close = (close / 4) + 50;



  sbi(PORTD, 1);        // starta främre sändaren
  delay_ms(500);
  delay_ms(500);

  for(int i = 0; i < 4; i++){ // Ta två värden och få ett
medelvärde
    ADMUX = 0x41;             // ställ in ADn till främre
mottagaren
    delay_ms(500);
    ADCSR = 0xC0;             // initiera ADn

    while((ADCSR && ADIF) != 1){}   // gör så att vi
väntar på att ADn är klar, maska med ADIF

    low = ADCL;
    high = ADCH;
    vect[i] = high * 256 + low;
  }
  cbi(PORTD, 1);        // stäng av främre sändaren
  far = vect[0] + vect[1] + vect[2] + vect[3];
  //far >> 2;     // skifta två steg (dela med fyra)
  far = (far / 4) + 45;
}

void checkAll(){        // kollar alla sidorna

  checkLeft();

  checkRight();
```

```
  checkFront();
}

void checkSides(){
  if(standing == 0){       // om vi står vända mot norr
    westWall  = checkLeft();
    northWall = checkFront();
    eastWall  = checkRight();

    labyrint[x][y][3] = westWall;
    if(x-1 > 0){
      labyrint[x-1][y][1] = westWall;
    }

    labyrint[x][y][0] = northWall;    // sätt den norra
väggen och nästa rutas södra vägg till northWall
    if(y-1 > 0){
      labyrint[x][y-1][2] = northWall;
    }

    labyrint[x][y][1] = eastWall;
    if(x+1 < 9){
      labyrint[x+1][y][3] = eastWall;
    }


  }else if(standing == 1){   // om vi står vända mot
öster
    northWall = checkLeft();
    eastWall  = checkFront();
    southWall = checkRight();

    labyrint[x][y][0] = northWall;
    if(y-1 > 0){
      labyrint[x][y-1][2] = northWall;
    }

    labyrint[x][y][1] = eastWall;
    if(x+1 < 9){
      labyrint[x+1][y][3] = eastWall;
    }

    labyrint[x][y][2] = southWall;
    if(y+1 < 12){
      labyrint[x][y+1][0] = southWall;
    }

  }else if(standing == 2){     // om vi står vända mot
söder
    eastWall  = checkLeft();
    southWall = checkFront();
    westWall  = checkRight();

    labyrint[x][y][1] = eastWall;
    if(x+1 < 9){
      labyrint[x+1][y][3] = eastWall;
    }

    labyrint[x][y][2] = southWall;
    if(y+1 < 12){
      labyrint[x][y+1][0] = southWall;
    }

    labyrint[x][y][3] = westWall;
    if(x-1 > 0){
      labyrint[x-1][y][1] = westWall;
    }

  }
```

```
  else if(standing == 3){    // om vi står vända mot
väster
    southWall = checkLeft();
    westWall  = checkFront();
    northWall = checkRight();

    labyrint[x][y][2] = southWall;
    if(y+1 < 12){
      labyrint[x][y+1][0] = southWall;
    }

    labyrint[x][y][3] = westWall;
    if(x-1 > 0){
      labyrint[x-1][y][1] = westWall;
    }

    labyrint[x][y][0] = northWall;
    if(y-1 > 0){
      labyrint[x][y-1][2] = northWall;
    }
  }
}

void dijkstras(){
}

// huvudprogram
void main(void){
  DDRA = 0xE0;
  DDRB = 0x3F;
  DDRC = 0x03;
  DDRD = 0x87; // Pekar ut inportar och utportar

  init();


  while(standing == 0 || standing == 2){
    checkSides();

  }

}

void delay_ms(unsigned short ms) {
  unsigned short outer1, outer2;
  outer1 = 100;         // 200 från början
  while (outer1) {
    outer2 = 50;   // från början, tror jag
    while (outer2) {
      while ( ms ) ms--;
      outer2--;
    }
    outer1--;
  }
}
```