2009

Av:
Aron Lidé – dt05al1
Marie Li Korse – dt05ml9

Handledare:
Bertil Lindvall

WALL E.
WALL EVADER
EDI021 – DIGITALA PROJEKT

Sammanfattning

Den här rapporten behandlar konstruktionen av en självgående robot som klarar av att undvika objekt framför den, samt därefter välja att fortsätta höger eller vänster beroende på vilken väg som är mest lämplig. Konstruktionen är uppbyggd av microkontrollern AVR ATMega32, en analog avståndssensor, en servo samt andra passande komponenter. Rapporten innehåller bland annat specifikationer för modulen, hårdvaruinformation, implementation, kopplingsschema samt en diskussion av resultatet och eventuell vidareutveckling.

Innehållsförteckning

Inledning

Detta projekt är genomfört i kursen Digitala Projekt – EDI021. En praktisk kurs som syftar till att ge elever möjlighet att designa och konstruera en prototyp av ett digitalt system. Man ges som student även en del övning i att planera sitt arbete väl och att producera ett resultat under viss tidspress.

Ursprungligen var tanken att skapa en robot som kunde söka av och spara ner sin omgivning för att sedan presentera den insamlade datan på en display. Idén ansågs dock innebära för mycket arbete i relation till tiden som var rimligt att spendera på kursen och blev därför omarbetad. Konstruktionen utgår dock ifrån originalidén och är därför enkel att vidareutveckla.

Arbetet med konstruktionen har gett oss en djupare förståelse för inbyggda system, hårdvara samt programmering av microkontrollers. Syftet med rapporten är att även kunna erbjuda andra den kunskap som erhållits av projektet.

Arbetsmetod

Eftersom ingen av oss haft särskilt stor erfarenhet av att arbeta med microkontrollers och hårdvara i största allmänhet, valde vi att arbeta på ett sätt som gjorde felsökning så enkel som möjligt. Det gick ut på att koppla in en komponent i taget, och köra ett program som var skrivit för att testa just den. Detta kom vi att ha stor nytta av i slutet av projektet när alla delar skulle integreras. Ifall det var någon komponent som inte längre fungerade som tänkt, kunde vi köra våra testfiler och jämföra med det slutgiltiga programmet.

Kravspecifikation

Arbetsgången som gällde i kursen var att ha höga målsättningar och väldigt optimistiska krav. Tanken var att man skulle ha något högt att sträva efter och skala ner projektet efterhand om så behövdes. Emellertid blev ändringarna i projektet så stora att kravspecifikationen gjordes om helt. Nedan presenteras därför båda kravspecifikationer.

Den ursprungliga kravspecifikationen

1. Modulen ska kunna avgöra vilken yta som den befinner sig på.
2. Modulen ska klara av att avgöra och registrera utifall en yta innehåller ett hinder eller är åtkomlig.
3. Modulen ska kunna avgöra om given yta tidigare är utforskad eller inte.
4. Modulen ska kunna utforska ett område inom en given ram.
5. Ett område ska registreras som fullständigt utforskat då all åtkomlig yta inom en given ram är registrerad som hinder eller fri yta.
6. Modulen ska kunna sättas ingång, stängas av, startas/startas om, pausas, fortsättas och visa hittills registrerad omgivning.
    6.1. Att sätta igång respektive stänga av modulen innebär att strömtillförseln sätts på respektive av.

6.2. Då modulen startas ges modulen en förutbestämd ram för vilket område som ska utforskas och en startposition. Allt område definieras som outforskat förutom ytan som täcks av modulen.

6.3. Då modulen pausas ska modulen stanna och utforskandet stoppas.

6.4. Då modulen fortsätter från pausat tillstånd ska utforskandet fortsättas.

6.5. Att visa omgivningen ska endast vara möjligt i pausat tillstånd. Endast en delmängd av omgivningen visas åt gången på skärmen. Användaren ska ha möjlighet att justera vilken delmängd av omgivningen visas via en knappsats.

7. Området på skärmen ska ha skiljda representationer av utforskad yta, fri yta samt hinderbelagd yta.

8. När en omgivning är fullständigt utforskad ska modulen pausas och indikera detta genom att en lysdiod slås på.

9. En yta representerar en area på 5x5 cm^2.

10. På skärmen ska varje yta representeras av en pixel.

Den nya kravspecifikationen

1. Modulen ska kunna sättas igång och stängas av.

2. Modulen ska kunna förflytta sig framåt, samt rotera cirka 90 grader i båda horisontella riktningar.

3. Modulen ska kunna mäta avståndet till hinder inom avståndet 20 – 150 cm i riktningarna rakt fram, åt höger samt åt vänster utifrån modulens perspektiv där rakt fram anses vara körriktningen.

4. Då modulen startas ska den köra rakt fram, till dess att ett framförliggande föremål i dess närhet registreras.

5. När ett föremål har registrerats ska modulen kontrollera avståndet till närmaste hinder åt vänster och höger, från konstruktionen sett.

6. Modulen ska efter en kontroll av avstånd åt höger och vänster, svänga åt det håll vars avstånd registrerat som längst till närmaste hinder.

7. Efter att modulen har svängt, ska samma programslinga exekveras som när den nyss har startats.

8. Konstruktionen ska hårdvarumässigt vara utformad på ett sådant sätt att den är enkel att vidareutveckla till en modul som uppfyller den ursprungliga kravspecifikationen.

Komponenter

De komponenter som användes för att bygga hårdvaran beskrivs ingående i denna sektion. Kopplingsschemat för hela modulen visas i Appendix, Figur 1. De komponenter som användes var:

- Microkontroller Atmel AVR ATmega32
- Long Distance Measuring Sensor GP2Y0A02YK
- Parallax Standard Servo (#900-00005)
- Full Bridge L298
- Octal 3-State Noninverting D-Latch 74HC373
- Ultra Low Dropout Regulator LP3855ET-5.0
- Två eldrivna motorer med hjul
- Strömbrytare
- Lysdioder och vanliga dioder
- Resistorer, kondensatorer och en spole
- Stödhjul

Microkontroller Atmel AVR ATmega32

ATMega32:an är grundpelaren för styrningen av modulen. Det är processorn som innehåller och exekverar programkoden som bestämmer vad de olika delarna av modulen ska göra. Den har 40 pinnar att koppla in på samt många inbyggda funktioner som har varit mycket användbara i vårt arbete, bl a A/D-omvandling och klockor. Den har också pinnar avsedda för inkoppling av JTAG, som är nödvändig för att kunna överföra programkod från datorn.

Long Distance Measuring Sensor GP2Y0A02YK

Detta är en avståndsmätare som hela tiden ger en analog signal som skiftas beroende på avståndet mellan mätaren och det den är riktad mot. Den mäter avstånd mellan 20-150 cm. Avståndsmätaren i sig var lätt att koppla in då den endast behövde matningsström (VCC och jord) och en pinne kopplad till en av pinnarna med A/D-omvandlare på microkontrollern. A/D-omvandlingen i microkontrollern var då en mycket större del av problemet, som tas upp i mjukvarudelen av rapporten.

Parallax Standard Servo (#900-00005)

Servomotorn användes för att ha avståndsmätaren på så att den kunde riktas i olika riktningar. Liksom med avståndsmätaren, var servomotorn lätt att koppla in, och arbetet låg än en gång i programmeringen av microkontrollern. Servon styrs utav pulser som skickas via en enda sladd som är inkopplad till microkontrollern, vilket både sparar pinnar och ger möjlighet till ett mycket högupplöst riktningsområde. För att få en stadig och enkelt programmerad styrning av servon kopplades den in på pinne PD5 (OC1A) som är en av pinnarna som är avsedda för utskickning av klocksignaler.

Full Bridge L298

L298:an användes för att styra strömmatningen till motorerna till hjulen. Eftersom spänningen faller så pass mycket över microkontrollern och för att den inte skulle klara av strömflödet som

krävs för att driva motorerna igenom den kan inte motorerna kopplas direkt till microkontrollern. Strömflödet behövdes också kunna ändra riktning då motorerna skulle gå åt andra hållet när modulen skulle svänga. Kort sagt fungerar L298:an så att signalerna som skickas in på dess insignalportar skickas ut på utsignalsporterna, men med en annan matningsspänning som kopplas in på en annan pinne på komponenten. Andra pinnar användes för att sätta på och stänga av styrningen från insignalportarna.

Octal 3-State Noninverting D-Latch 74HC373

Denna komponenten tar 8 insignaler och skickar vidare dem till utportarna om komponenten är aktiverad. Den var i själva verket onödig att använda med vår nuvarande design, men i vår ursprungliga ritning var den nödvändig för att kunna använda som ett mellansteg mellan microkontrollern och ouputkomponenter som skulle kopplas in på de 8 pinnarna som användes som en buss, eftersom komponenten kan aktiveras och inaktiveras vilket leder till att man kan bestämma vilken komponent som är inkopplad till bussen för tillfället. En likadan komponent har kopplats till bussen, men på andra hållet så att den fungerar som en latch för inputsignaler till microkontrollern (syns inte på kretsschemat). Den var från början avsedd till att koppla in knapparna på, men är för tillfället oanvänd, och vid möjlig enkel utbyggnad för att t ex göra modulen sladdstyrd med knappar skulle den troligen inte användas ändå då det är bättre att använda andra frigjorda pinnar.

Ultra Low Dropout Regulator LP3855ET-5.0

Eftersom vi bara kunde få tag på ett batteri på 6V så var vi tvugna att använda en spänningsregulator för att minska spänningen över komponenterna, som för det mesta ska ha 5V över sig. Några resistorer och kondensatorer var nödvändiga att kopplas in för att få komponenten att fungera korrekt.

Mjukvara

Mjukvaran består av en enda klass som har en oändlig while-loop. Den börjar med en while-loop som skickar signaler till L298:an att aktivera strömmatningen till motorerna en väldigt kort stund och sedan läses avståndsmätaren av och kollar ifall det finns något framför modulen inom tillräckligt kort avstånd för att den ska stanna. Om det finns det lämnas while-loopen, annars fortsätter modulen att köra framåt.

Sedan startas servon och pulser med rätt längd skickas ut till den för att den ska svänga till höger. Programmet pausas en stund för att låta servon hinna vrida sig till sin position innan avståndsmätaren läses av igen. Sedan görs samma sak för att kolla avståndet på vänster sida. Efter det vrids servon igen så att avståndsmätaren pekar framåt igen. Avståndsvärdena till höger och vänster jämförs och programmet går in i en sats för att svänga åt det hållet som har mest utrymme. Där ändras insignalerna till L298:an för att få ena motorn att gå baklänges, sedan skickas korta impulser till L298:an för att aktivera strömmatningen till motorerna under ett antal korta sekvenser för att få modulen att svänga. Därefter stängs servon av igen, för att spara ström, innan programmet återvänder till början av while-loopen.

A/D-omvandling

Koden för A/D-omvandlingen består av inställningar som skrivs till ADMUX-registret i microkontrollern. Där väljs inputpinne, vänster- eller högerjustering av de 10 bitarna som A/D-

omvandlingen ger tillbaka samt en spänningsreferens för det högsta spänningsvärdet på omvandlingsområdet (det lägsta är 0V). Den interna spänningsreferensen på 2,56V valdes för att få så stor upplösning som möjligt, eftersom den högsta spänningen avståndsmätaren ger tillbaka är 2,3V. För att starta en omvandling sätts de två första bitarna i registret ADCSRA till höga, sedan läses ADCL och ADCH av och slås ihop för att få det korrekta värdet.

Timer

Den svåraste biten av mjukvaran låg i att få pulserna till servon att vara korrekta. Servon styrs av en PWM-signal (Pulse Width Modulation) som är en signal med fast frekvens, men där längden förändras. Frekvensen behöver vara 50Hz, dvs att en puls var 20:e ms skickas. Vi behöver alltså sätta både när en ny puls börjar och när den slutar. Det första värdet är konstant och ändras aldrig medan det andra är det som ändras på för att styra servon.

Klockan Timer1 används och dess frekvens bestäms av värdet på registret ICR1. Med hjälp av en formel i microkontrollerns datablad fastställs det att eftersom systemklockans frekvens är 8MHz, bör prescalern sättas till 8 och ICR1 till 10000 för att få en 50Hz-signal (8MHz / (2*8*10000) = 50Hz). Längden på pulsen bästäms av värdet som sätts till registret OCR1A. Vi skiftade mellan tre pulslängder för våra positioner (70 = höger, 600 = mitten, 1000 = vänster). Signalen skickas ut på pinne OC1A, vilket är den som servon är inkopplad på.

Resultat

Konstruktionen resulterade i en välfungerande prototyp som uppfyllde samtliga nya krav. Modulen kunde dock blivit mer stabil om mer tid lagts på finjusteringar och kalibrering.

Problem och svårigheter

Prototypen är inte riktigt så stabil som man hade kunnat önska. Exempelvis drar den snett åt vänster då den ska köra rakt fram och den svänger inte exakt 90 grader, det vill säga rakt åt höger eller rakt åt vänster, då den ska svänga. Vidare lyckades vi inte få servon som avståndssensorn sitter på att vrida sig till dess båda extremlägen, vilket gör att avståndsmätningen inte heller sker i helt rätt vinkel. Servon började nämligen vackla fram och tillbaka väldigt snabbt eller fastnade helt när vi valde att ha den i något extremläge. Enligt specifikationen skulle den klara av lite mer vridning än vad som utförs i dagsläget, men vi ansåg att det nästbästa var fullt dugligt också.

Andra svårigheter som stöttes på under projektet var att få rätt på samtliga register som skulle aktiveras för att få timern att fungera. Även detta löste sig dock tillslut.

Lärdomar

Under detta projekt har vi fått lära oss mycket om hårdvara och en hel del om hur man söker effektivt bland datablad samt att förstå och ta till sig informationen de ger. Dessutom har man fått lära sig hur man programmerar en microkontroller, och med hjälp av denna styra andra komponenter. Vi har även fått erfarenhet av att planera om ett projekt för att hinna prestera ett resultat inom en viss deadline.

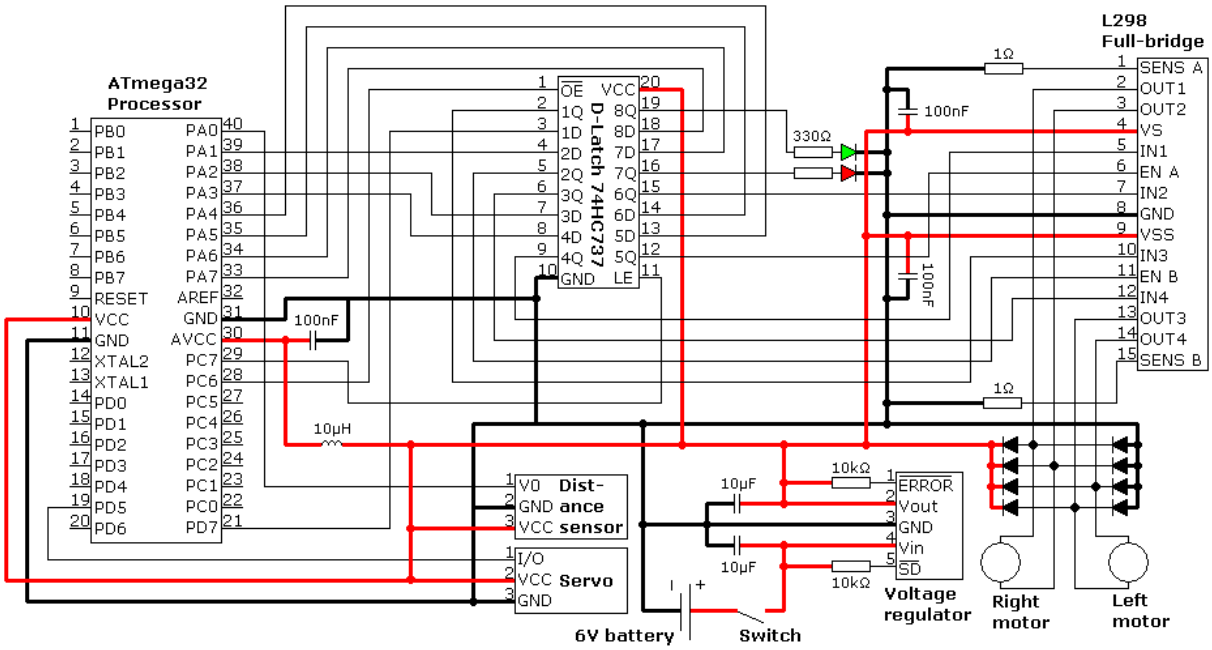
Vidareutveckling och förbättringar

Som tidigare nämnts var konstruktionen från början tänkt kunna göra något mer än det den för tillfället klarar av. Nämligen att kartlägga området den kör på och visa upp detta på en display. Sättet för att gå till väga implementationsmässigt hade varit dela upp området som ska köras på i ett rutnät och spara undan information om varje ruta i en matris internt. Varje element i matrisen hade kunnat lagra en två bitar stor variabel där en siffra motsvarade outforskat område, en för utforskat och fritt samt en för utforskat med hinder på. Informationen hade enkelt kunnats få med hjälp av befintlig hårdvara. Däremot skulle ett externt minne krävas för att lagra matrisen om man skulle vilja ha relativt hög upplösning, då minnet i mircokontrollern inte hade räckt så långt. Vidare hade en display behövts koppla in om man skulle vilja presentera datan på ett visuellt sätt, och gärna en knappsats för att kunna välja vilket område av kartan som skulle visats.

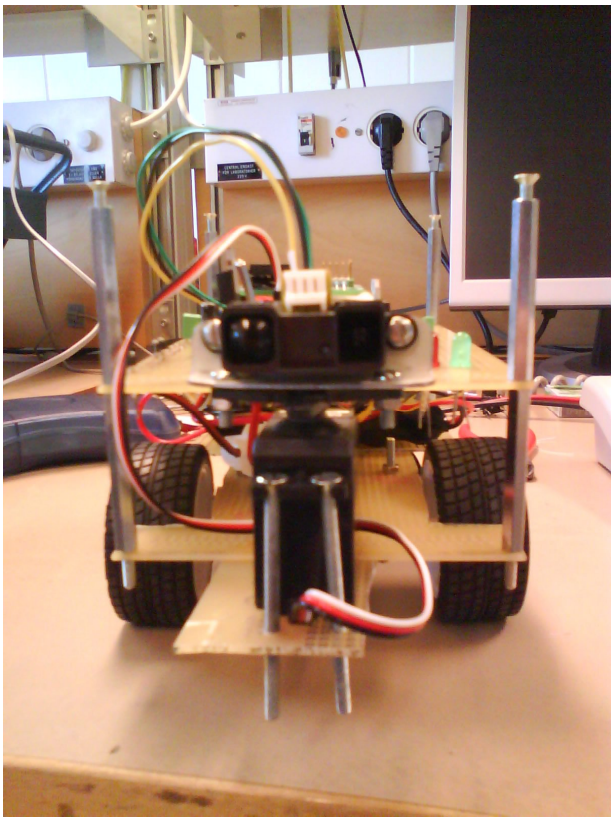
Vår konstruktion är utformad så att den nya hårdvaran som hade krävts för den här typen av vidareutveckling är enkel att göra. Outputen från processorn är nämligen kopplad via en D-latch som går att aktivera via porten chipselect. Detta innebär att man inte behöver använda fler pinnar från processorn för att koppla in en knappsats eller display, utan kan utnyttja samma pinnar som används till D-latchen. Andra sätt som konstruktionen skulle kunna vidareutvecklas på är exempelvis att fästa sensorer på undersidan av främre delen av den, så att kanter hade kunnat upptäckas.

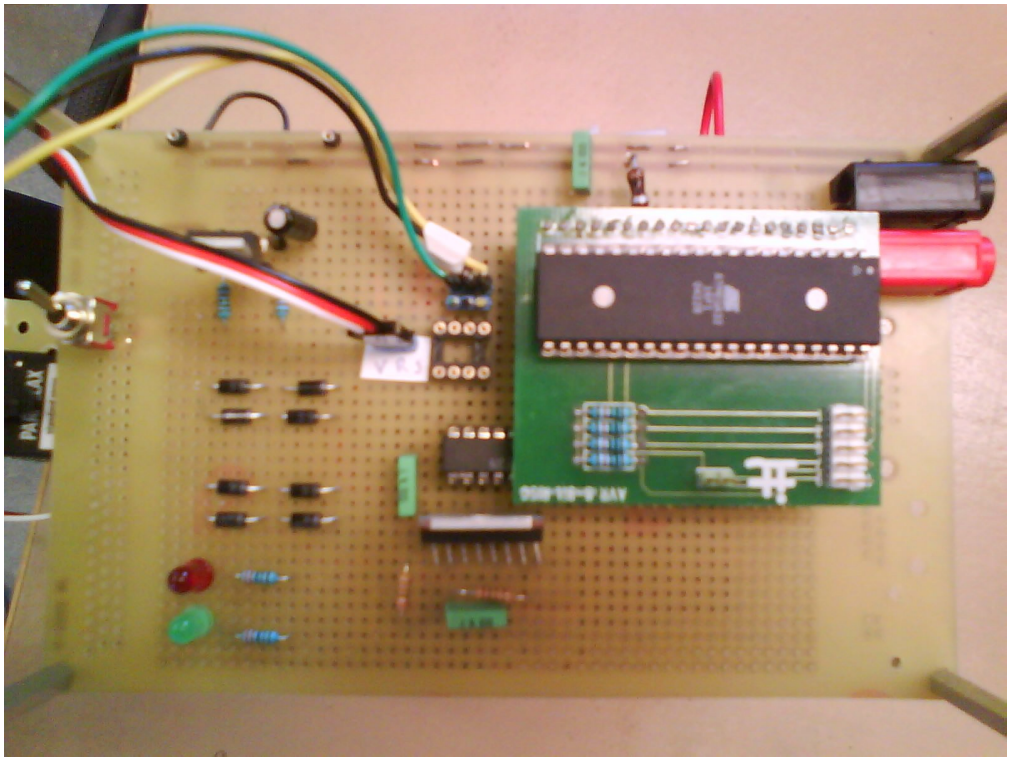
Någonting som definitivt hade kunnat förbättras, och garanterat hade behövts om konstruktionen gjorts mer komplicerad är refaktoriseringar och bättre struktur av koden. Tanken var först att få allting att fungera på enklast möjliga vis, vilket ledde till en del klipp och klistra kod. Dock valde vi att presentera den kod som faktiskt ligger på konstruktionen i dagsläget som vi till hundra procent vet fungerar.

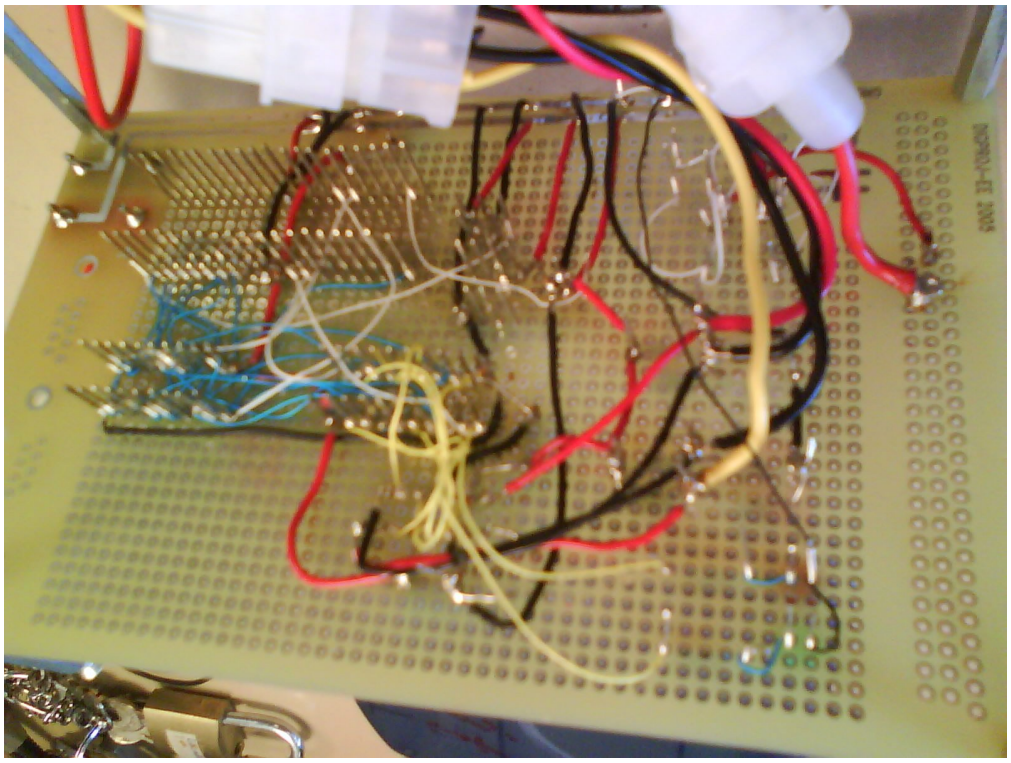
Appendix A : Figurer



Figur 1: Blockschema







Appendix B: Källkod

```
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
        int anhigh = 0;
        int anlow = 0;
        int antot = 0;
        int go = 1;
        int green = 1;
        int left = 0;
        int right = 0;
        int speed = 200;

        // set a1 to a7 to output
        DDRA = 0b11111110;

        // two first bits, voltage reference internal
        // bit 3, output bits left adjusted
        // the rest bits, input pin is ADC0
        ADMUX  = 0b11000000;


        // Prescaler N=8 then TOP(ICR1) = 10000
        // Because 8MHz / (2*8*10000) = 50Hz
        TCCR1B = 0b00000010;
        ICR1 = 10000;

        //8-bit, Inverted PWM
        TCCR1A = 0b10000011;

        // Adjust servo to pointing forward
        DDRD = 0b10100011;
        OCR1A = 600;
        _delay_ms(2000);
        DDRD = 0b10000011;

        PORTD = 0b00100011;
        // set c6 and c7 to output (OE, LE for outputlatch)
        DDRC = 0b11000000;
        // set (pin c7)LE high and (pin c6)OE low (output = input)
        PORTC = 0b10000000;

        // In2, In4 set to on, enable A and B on (pin A4 and A1)
        PORTA = 0b10110110;
        // Turn off enable A and B, wheels should stop spinning
        PORTA = 0b10001000;

        // drive forward until the distance sensor registers a close object
        ADCSRA = 0b11000000;
        PORTA = 0b10110110;
        _delay_ms(100);
        PORTA = 0b10001000;
        _delay_ms(400);
        anlow = ADCL;
        anhigh = ADCH;
        antot = anlow + 256*anhigh;
        while(1) {
```

```
while(go){

        ADCSRA = 0b11000000;
        PORTA = 0b10110110;

        _delay_ms(speed);
        PORTA = 0b10001000;
        _delay_ms(400);
        anlow = ADCL;
        anhigh = ADCH;
        antot = anlow + 256*anhigh;

        if(antot > 900 && antot != 1023){
                go = 0;
        }
}

go = 1;
PORTA = 0b11001000;

// Start the servo
DDRD = 0b10100011;
OCR1A = 600;
_delay_ms(50000);

// Turn the servo right, and let the distance sensor read and
// store the value
OCR1A = 70;
_delay_ms(50000);
ADCSRA = 0b11000000;
_delay_ms(10);
anlow = ADCL;
anhigh = ADCH;
right = anlow + 256*anhigh;

OCR1A = 600;
_delay_ms(50000);

// Turn the servo left, and let the distance sensor read and
// store the value
OCR1A = 1000;
_delay_ms(50000);
ADCSRA = 0b11000000;
_delay_ms(10);
anlow = ADCL;
anhigh = ADCH;
left = anlow + 256*anhigh;

// Turn the servo straight forward
OCR1A = 600;
_delay_ms(50000);

if(left < right) {
        // Turn the entire construction to the left
        PORTA = 0b10001000;
        for(int i = 0; i < 25; i++){
                ADCSRA = 0b11000000;
                PORTA = 0b10110010;
                PORTD = 0b10100011;
                _delay_ms(speed);
                PORTA = 0b10001000;
                _delay_ms(400);
                anlow = ADCL;
```

```
                        anhigh = ADCH;
                        antot = anlow + 256*anhigh;
                    }

                    PORTD = 0b00100011;


                }
                else {
                    //Turn the entire construction to the right
                    PORTA = 0b01001000;
                    for(int i = 0; i < 25; i++){
                        ADCSRA = 0b11000000;
                        PORTA = 0b10011110;
                        _delay_ms(speed);
                        PORTA = 0b10001000;
                        _delay_ms(400);
                        anlow = ADCL;
                        anhigh = ADCH;
                        antot = anlow + 256*anhigh;

                    }

                }
                // Turn off the servo
                DDRD = 0b10000011;
        }
        return 1;
}
```