LOOKY LUKE

- Att välja en lyckosam väg

EITF11 – Digitala projekt

Caroline Hellström och Ville Orlander Arvola
Industriell ekonomi I10

Handledare: Bertil Lindvall

2013-05-19

Abstract

This project was done within the course EITF11 Digitala projekt at Lunds Tekniska Högskola, spring 2013. The course aims to teach how an industrial development project is executed. A prototype was created, programmed and tested.

In this project an automotive vehicle which could avoid obstacles was built. A requirements specification was first written and approved by the mentor. The different components were then connected according to their datasheets. The foundation of the construction was an ATMega16 microcontroller. A servo was connected with a distance measuring sensor to turn the sensor in different directions. The sensor was used to detect obstacles in front of the vehicle. To move the vehicle caterpillar treads were used together with two engines.

When turned on, the vehicle moves forward and continuously measures the distance to obstacles in front of it. When the distance reaches a critical level the vehicle stops. It then measures to the left and to the right, to determine a suitable new direction. The vehicle then reverses, makes a turn and continues driving forward in the new direction until another obstacle is detected.

It has been a challenging and educational project. Especially in terms of working within unfamiliar areas and dealing with uncertainty.

Innehållsförteckning

1 Inledning

Projektuppgiften inom kursen Digitala Projekt (EITF11) innefattade konstruktion av en prototyp. Prototypen skulle sedan byggas, programmeras i C samt testas. Samtidigt skulle lämplig dokumentation göras så att prototypen kan återskapas och/eller vidareutvecklas.

I den här rapporten beskrivs arbetet med att utveckla ett självgående fordon som ska kunna undvika hinder. Fordonet ska köra framåt tills den stöter på ett hinder. Den ska då stanna och avgöra ny lämplig riktning, för att sedan fortsätta köra i denna riktning. Som grund för konstruktionen tillhandahölls en mikroprocessor, ATMega16.

2 Kravspecifikation

2.1 Grundläggande krav

Nedan följer kraven på de grundläggande egenskaperna prototypen ska uppfylla. Fordonet ska:

- Kunna köra framåt, bakåt, svänga höger och vänster.
- Kunna läsa av sin omgivning och undvika hinder genom att avgöra ny lämplig riktning att fortsätta köra i.
- Fordonet ska vara batteridrivet.

2.2 Utökade krav

De utökade kraven tas till hänsyn i mån av tid. Nedan följer kraven på de utökade egenskaperna prototypen i mån av tid ska uppfylla. Fordonet ska:

- Signalera med ljus vid hinder, till exempel blinkers.
- Kunna koppla ifrån sensorn och fjärrstyras.

3 Teori

3.1 Hårdvara

Nedan beskrivs de hårdvarukomponenter som användes i prototypen.

3.1.1 Mikroprocessor – ATMega16

ATMega16 är en 8-bits mikroprocessor med 16kB programmerbart flash-minne som programmeras i C. Processorn har 32 programmerbara I/O-pinnar och ett JTAG interface för debugging. Mikroprocessorn innehåller även AD-omvandlare samt tre timers/counters med möjlighet till både interna och externa avbrott. Driftspänningen är 4,5 – 5,5 V.

3.1.2 Servomotor – Parallax #900-00005

Parallax-servon är en servomotor som genom pulsvågsmodulation kan hålla positioner mellan 0 och 180 grader. Servot kräver pulser var 20:e ms, där positionen för servot avgörs av pulslängden (0,75 – 2,25 ms). Driftspänningen är 4 - 6 V.

3.1.3 Avståndsmätare – SHARP GP2Y0A02YK

Avståndsmätaren kan genom triangulering via sin sensor mäta avståndet till objekt på 20 till 150 centimeters avstånd från sensorn. Avståndsmätaren mäter avstånd genom att skicka ut en IR-stråle som reflekteras på objektet framför. Avståndet avspeglas i spänningen på den analoga signal som

sensorn skickar ut. Ju längre från avståndsmätaren objektet befinner sig, desto lägre blir spänningen. Driftstpänningen är 4,5 – 5,5 V.

3.1.4 H-brygga – Dual Full-Bridge Driver L298

Dual Full-Bridge Driver L298 innehåller två H-bryggor som används för att ändra riktningen på likström. Dessa kan användas för att styra exempelvis motorer. Den klarar av en driftsspänning på upp till 46 V.

3.1.5 Spänningsregulator – LP3855

LP3855 reglerar spänningen ut till en konstant spänning, exempelvis från 6 V på ett batteri till 5 V som mikroprocessorn ATMega16 behöver. Driftspänningen är 2,5 – 7,0 V.

3.1.6 Larvfötter

Larvfötter med tillhörande likströmsmotorer. Driftspänningen är 6 V.

3.1.7 Lysdioder

Röd, gul och grön lysdiod med driftspänning 2 V.

3.1.8 Strömbrytare

Enkel på/av-strömbrytare.

3.1.9 Batteri

Laddningsbart batteri med en spänning på 6 V.

3.2 Mjukvara

Nedan beskrivs den mjukvara som användes i projektet.

3.2.1 PowerLogic

PowerLogic är en mjukvara som kan användas för att rita allt från enkla till avancerade kopplingsscheman.

3.2.2 AVR Studio 4

AVR Studio 4 är en integrerad utvecklingsplattform för utveckling och debugging av bland annat AVR-mikroprocessorer. I AVR Studio 4-miljön skrivs, byggs och debuggas applikationer i C/C++ eller assemblerkod.

4 Metod

4.1 Övergripande

Första delen av projektet bestod i att skriva en kravspecifikation (se 2 Kravspecifikation) som i korta drag beskrev konstruktionens tänkta egenskaper. Ur kravspecifikationen kunde de olika komponenterna som skulle ingå i konstruktionen sedan härledas. Med hjälp av datablad för respektive komponenter togs ett kopplingsschema fram som ritades med hjälp av programmet PowerLogic. Se Bilaga 1.

När kopplingsschemat var godkänt av handledare började konstruktionsdelen av projektet. Förutom de grundläggande komponenterna beskrivna i avsnitt 3.1 användes resistorer, kondensatorer, sladdar, lödtenn, skruvar, muttrar samt ett tomt kretskort som utgjorde basen för elektroniken. De

elektriska komponenterna löddes eller virades ihop enligt kopplingsschemat. Kretskortet skruvades sedan fast på larvfötterna.

Då alla komponenter var påmonterade kopplades konstruktionen till en 5 V-strömkälla samt en dator genom JTAG-interfacet. I AVR Studio 4 programmerades och felsöktes det tills fordonet fick önskat beteende. Även korrigeringar i hårdvaran fick göras allt eftersom fel upptäcktes.

4.2 Beskrivet utifrån nyckelkomponenter

Spänningsregulatorn

Fordonet ska drivas av ett 6 V-batteri. Mikroprocessorn och avståndsmätaren behöver däremot en spänning på 5 V, varför en spänningsregulator måste användas för att omvandla spänningen till 5 V för dessa komponenter.

Larvfötterna

För att kunna driva larvfötternas två 6 V-motorer samt kunna låta dessa byta riktning behövs H-bryggor. Genom dessa kan de två motorerna styras separat och köras i båda riktningar med hjälp av mikroprocessorn. För att bestämma hastigheten på motorerna används pulsbreddsmodulering.

Avståndsmätaren

Då fordonet ska kunna upptäcka hinder under tiden den kör mäter avståndsmätaren kontinuerligt sin omgivning. För att kunna använda resultatet från avståndsmätaren måste den analoga utsignalen från IR-sensorn först omvandlas till digital. Detta görs genom att använda mikroprocessorns interna AD-omvandlare. För att få maximal upplösning på AD-omvandligen behövs en inputklockfrekvens mellan 50kHz och 200kHz. Då mikroprocessorn är inställd att arbeta på 8MHz väljs 128 som prescaler, vilket ger en inputfrekvens på 62,5 kHz. För att avläsa resultaten av AD-omvandlingarna används avbrott.

Servot

För att ställa in servot i rätt läge behövs pulser på 0,75-2,25 ms i internvall om 20 ms. För att åstadkomma dessa används mikroprocessorns interna timer. Genom att välja 8 som prescaler och låta mikroprocessorn generera avbrott var 100:e klockslag görs avbrott varje 0,1 ms då mikroprocessorn är inställd på 8 MHz. För att få intervall om 20 ms låter man en count-variabel öka ett steg vid varje avbrott, och låta den gå i cykler från 0 till 200. Med hjälp av denna count-variabel kan pulser skickas till servot för att ställa denna i önskat läge.

5 Resultat

Den färdiga prototypen fungerar i enlighet med kravspecifikationen. Av de utökade kraven införlivades den om signalering, medan fjärrstyrningen uteblev på grund av tidsbrist.

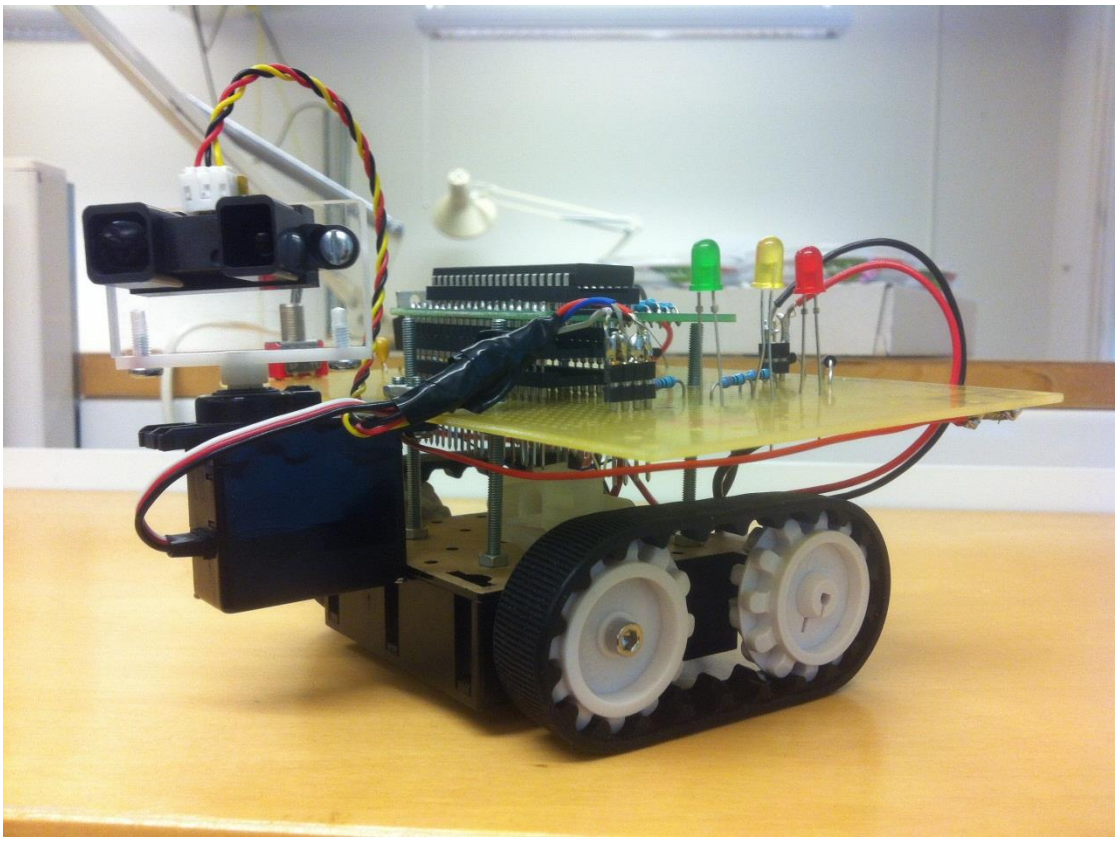


Bild 1. Färdig konstruktion.

6 Diskussion och analys

Projektet har varit roligt, utmanande och lärorikt. Vi har fått angripa problem inom områden som vi tidigare saknade erfarenhet av. Vi har aldrig tidigare arbetat med elektroniska komponenter på det här sättet, och vi har tidigare inte tittat på kopplingen mellan just hårdvara och mjukvara. Vidare har vi under projektet programmerat i C, vilket är nytt för oss. Instruktionerna för projektet var vaga, och stort ansvar lades på projektgrupperna att själva bestämma upplägg och arbetsmetoder. Generellt har projektet präglats av osäkerhet. En osäkerhet som man har blivit tvungen att lära sig hantera.

Vi valde att göra ett självgående fordon eftersom det verkade som ett kul projekt, samtidigt som resultatet är väldigt konkret. Man får helt enkelt se fordonet röra sig, servot vridas, lampor tändas/släckas etc. Under projektets gång stötte vi dock på flertalet utmaningar.

Den första utmaningen bestod av att bekanta sig med facktermer och förstå hur datablad används, för att därefter förstå hur de olika komponenterna skulle kopplas ihop. Så fort kopplingsschemat var klart gick det relativt snabbt att koppla ihop och bygga fordonet.

Då fordonet var färdigbyggt ägnade vi en del tid åt att testa de enskilda komponenterna genom att manuellt slå på och av portar genom AVR Studio:s debug-läge. Därefter var det dags att börja programmera.

Vi ägnade mycket tid åt att få servot att fungera. Här handlade det framför allt om att lära sig hur mikroprocessorns interna timer fungerar, samt vad avbrott innebär och hur dessa kan utnyttjas. Grundfaktan fanns i databladen, men för att förstå hur man skulle applicera denna behövdes ytterligare hjälp. Genom mycket googlande och en del handledning lyckades vi till slut få servot att fungera.

Något som försvårade programmeringen av servot var att vi under våra avbrott utförde för processorkrävande kommandon utan att vi tänkte på det. Vi hade helt enkelt gjort misstaget att tänka på tidskomplexitet som vi tidigare gjort vid exempelvis Java-programmering, utan att reflektera över vad ett enskilt kommando faktiskt innebär för en mikroprocessor.

Nästa stora utmaning låg i att förstå hur avståndsmätaren fungerade. Efter att ha ägnat en stor del av tiden åt att ha felsökt i koden upptäckte vi att det inte var koden det var fel på, utan att felet helt enkelt berodde på glapp i en av kontakterna. Då glappet var åtgärdat fungerade avståndsmätaren utan problem. Det var dock en lång process, med många kreativa lösningar på eventuella felkällor innan vi kunde isolera felet.

Mot slutet av projektet hade vi lyckats programmera de enskilda komponenterna var för sig, men det blev fel när vi försökte integrera dessa. Ett stort genombrott kom här då det visade sig att mikroprocessorn var inställd på att optimera den kod vi skrivit. Detta innebar i sin tur att processorn helt enkelt hoppade över vissa kodrader. Då vi tog bort den här ”optimeringen” fungerade plötsligt allt som tänkt.

Vår sista utmaning låg i att mikroprocessorn inte betedde sig likadant då den var inkopplad med JTAG som då den var frånkopplad. Genom att lägga till några kondensatorer kunde dock även det här felet åtgärdas.

7 Referenslista

7.1 Datablad

ATMega16:
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf

Dual Full-Bridge Driver:
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Analog/linear/l298.pdf

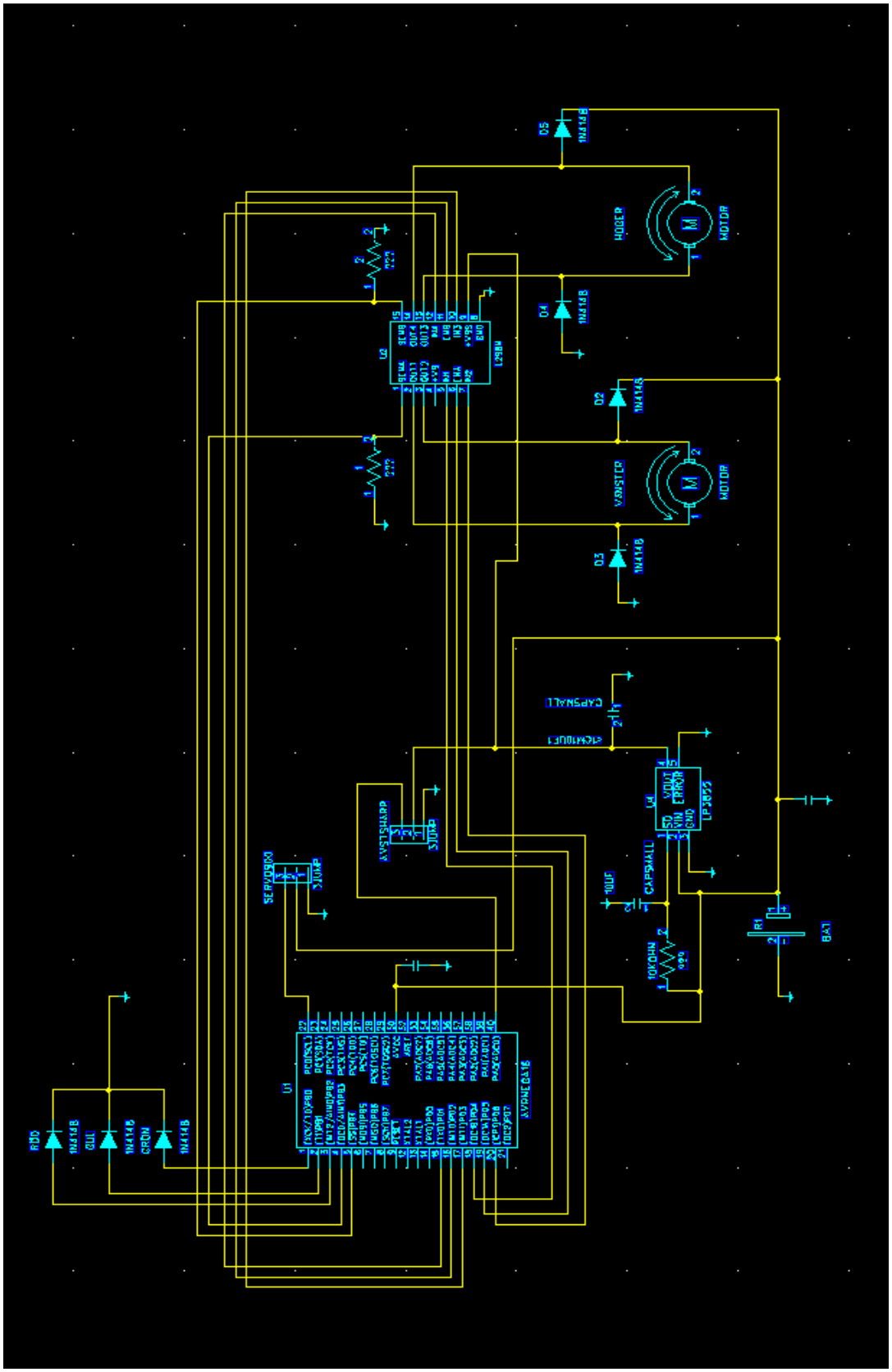
LP3855:
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Analog/voltage/lp3855.pdf

SHARP GP2Y0A02YK:
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Sensors/gp2y0a02.pdf

Parallax #900-00005:
http://www.parallax.com/Portals/0/Downloads/docs/prod/motors/900-00005-StdServo-v2.2.pdf

8 Bilagor

8.1 Bilaga 1 – Kopplingsschema



8.2 Bilaga 2 - Källkod

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

uint8_t stopDist, dist1, dist2, leftDist, rightDist, count, leftDir,
rightDir, midDir, servoDir, nextAction, leftE, rightE;
int dist, measureTime, reverseTime, turningTime;
long pauseTime;
_Bool obstacle, nextDist;

void main(void) {
          setup();
          wait(15000);
          forwardEngines();
          PORTB = 0x01; //Green light
          while (1) {
                    if (pauseTime > 0 ) {
          continue;
                    }

                    if (obstacle) {
                              PORTB = 0x04; //Red light
                              stopEngines();
                              scout();
                              PORTB = 0x02; //Yellow light
                              changeDirection(leftDist <= rightDist);
                              PORTB = 0x01; //Green light
                              forwardEngines();
                              obstacle = 0;
                    }
          }
}

void setup() {
          setupPorts();
          setupTimer();
          setupADC();
          setupVariables();
          sei(); //Enable global interrupts, used by timer and ADC

}

void setupPorts() {
          DDRD = 0xFF; //Configure Port D as output
          DDRB = 0xFF; //Configure Port B as output
          DDRA = 0x00; //Configure Port A as input
          DDRC |= 1 << PORTC0; //Configure PORTC0 as output
}

void setupTimer() {
          TCCR0 = 0x02; //Prescaler = 8
          TCNT0 = 0x00; //Start count = 0
          TIMSK = 0x02; //Enable Timer/Counter0 Overflow Interrupt
```

```
        OCR0 = 100; //Max count
}

void setupADC() {
        //8 000 000/50 000 = 160, 8 000 000 / 200 000 = 40 =>
choose prescaler: 128
        ADCSRA |= 1 << ADPS2;   //Prescaler: 128
        ADCSRA |= 1 << ADPS1;   //Prescaler: 128
        ADCSRA |= 1 << ADPS0;   //Prescaler: 128
        ADMUX = 0x60; //internal voltage, left adjusted, use ADC0
        ADCSRA |= 1 << ADIE;
        ADCSRA |= 1 << ADEN;
        ADCSRA |= 1 << ADSC; //Start next conversion
}

void setupVariables() {
        count = 0;
        pauseTime = 0;

        midDir = 13;
        leftDir = 18;
        rightDir = 8;
        servoDir = midDir;

        dist = 0;
        dist1 = 0;
        dist2 = 0;
        leftDist = 0;
        rightDist = 0;
        stopDist = 120; //Bigger value => shorter distance.
Default stopDist = 130

        nextAction = 1;
        nextDist = 1;
        obstacle = 0;

        reverseTime = 3000; //Default reverseTime = 5000
        measureTime = 5000; //Default measureTime = 7500
        turningTime = 6500; //Default turningTime = 6500

        leftE = 128; //Default leftE = 68
        rightE = 98; //Default rightE = 60
}

void wait(int time) {
        pauseTime = time;
        while (1) {
                if (pauseTime <= 0) {
                        return;
                }
        }
}

void scout() {
        changeSensDirection(rightDir);
        rightDist = dist;
```

```
          changeSensDirection(leftDir);
          leftDist = dist;
          changeSensDirection(midDir);
}

void changeSensDirection(int dir) {
          servoDir = dir;
          wait(measureTime);
}

void changeDirection(_Bool dir) { //true = turn left, false = turn
right
          reverseEngines();
          wait(reverseTime);
          stopEngines();
          wait(measureTime);
          leftE = 90;
          rightE = 82;
          if (dir) {
                    turnLeft();
          } else {
                    turnRight();
          }
          wait(turningTime);
          leftE = 128;
          rightE = 98;
          stopEngines();
          wait(measureTime);
}

void forwardEngines() {
          PORTD = 0x3C;
}

void reverseEngines() {
          wait(measureTime);
          PORTD = 0x66;
}

void stopEngines() {
          PORTD = 0x00;
}

void turnLeft() {
          rightEngine();
          reverseLeftEngine();
}

void turnRight() {
          leftEngine();
          reverseRightEngine();
}

void rightEngine() {
          PORTD &= ~(1 << PORTD1);
          PORTD |= 1 << PORTD2;
```

```
        PORTD |= 1 << PORTD3;
}

void leftEngine() {
        PORTD |= 1 << PORTD4;
        PORTD |= 1 << PORTD5;
        PORTD &= ~(1 << PORTD6);
}

void reverseLeftEngine() {
        PORTD &= ~(1 << PORTD4);
        PORTD |= 1 << PORTD5;
        PORTD |= 1 << PORTD6;
}

void reverseRightEngine() {
        PORTD |= 1 << PORTD1;
        PORTD |= 1 << PORTD2;
        PORTD &= ~(1 << PORTD3);
}




//Timer0 overflows every 0.1 ms
ISR(TIMER0_COMP_vect) {

        //Update counters
        count++;
        if (count == 200) {
                count = 0;
        }
        if (pauseTime > 0) {
                pauseTime--;
        }

        //Sensor position
        if (count == 0) {
                PORTC |= 1 << PORTC0;
        }

        if (count == servoDir) {
                PORTC &= ~(1 << PORTC0);
        }

        //Engine power
        if (count == 0) {
                PORTD |= 1 << PORTD2;
                PORTD |= 1 << PORTD5;
        }
        if (count == rightE) {
        PORTD &= ~(1 << PORTD2);
        }
        if (count == leftE) {
                PORTD &= ~(1 << PORTD5);
```

```
        }

        TCNT0 = 0x00; //Resets timer count
}

//Handle conversion
ISR(ADC_vect) {
        //Get average distance
        if (nextDist) {
                dist1 = ADCH;
                nextDist = 0;
        } else {
                dist2 = ADCH;
                dist = (dist1 + dist2);
                dist = dist >> 1;
                nextDist = 1;
                if (dist > stopDist) {
                        obstacle = 1;
                }
        }
        ADCSRA |= 1 << ADSC; //Start next conversion
}
//Catch unintended interrupts
ISR(BADISR_vect) {
}
```