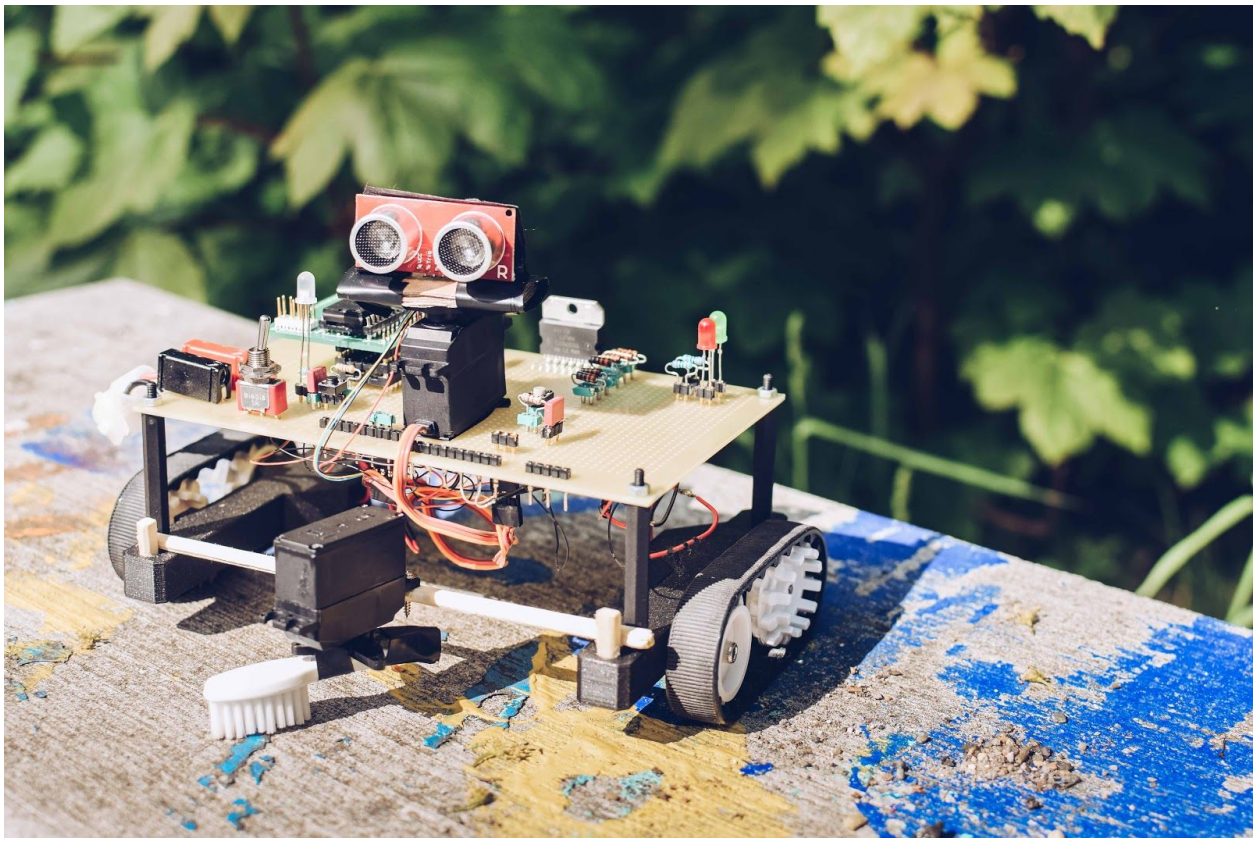
Little BrushBot



Digitala projekt, EITF12
Lunds tekniska högskola
Handledare: Christoffer Cederberg
Kursansvarig: Bertil Lindvall

August Asplund, August Lidfeldt & Emma Petersén
Maj, 2019

Abstract

The purpose of this project was to build a prototype from scratch. The project group decided to build an autonomous robot, Little BrushBot, with the feature that the robot should be able to brush the floor. Furthermore, Little BrushBot should be able to avoid obstacles appearing 15-50 centimeters in front of the robot.

The first step of the project was to specify requirements for the prototype. The next step was to draw a wiring diagram, put all the components together and at last, develop the software. During the project, the project members acquired knowledge in building an electric device and programming in C.

Innehållsförteckning

1 Inledning

1.1 Syfte

I kursen Digitala projekt är syftet att konstruera en prototyp från grunden för att på så sätt erhålla fördjupade kunskaper inom både hård- och mjukvara. Studenter i grupper om 3 studenter får under 8 veckors tid arbeta fram en idé till prototyp genom att exempelvis bekanta sig med datablad och på så sätt skapa ett kopplingschema, programmera och montera hårdvara. I slutet av projektet redovisas prototyp tillsammans med denna rapport, en muntlig redovisning och övrig dokumentation. Denna grupp har valt att utveckla en självkörande robot som ska kunna sopa golv med en liten borste.

1.2 Metod

Denna grupp valde att ta tillvara på en långtgående dröm för samtliga deltagare - att tillverka en självkörande robot. Allt eftersom tiden gått har robotens karaktär utvecklats och en kravspecifikation har upprättats, något som kommer att redovisas mer utförligt senare. Arbetet startade med att specificera samtliga krav som skulle finnas på roboten. Utifrån det diskuterades med handledarna vilka komponenter som skulle behövas. Genom att studera dessa komponenters datablad ritades ett kopplingsschema för roboten och utvecklandet av prototypen kunde ta start. Under utvecklandet har man kunnat koda för att kontinuerligt testa komponenter men mot slutet av hårdvaruutvecklingen tog den riktiga mjukvaruutvecklingen fart. Efter att all mjukvaruutveckling var klar, testades roboten för att lösa eventuella buggar.

2 Teori

Visionen för roboten är en självkörande robot som kan köra runt fritt i ett rum. Den ska kunna väja för hinder samt hjälpa till att sopa bort smuts på golvet, med hjälp av en liten borste. Idéen till Little BrushBot uppkom då en av gruppens medlemmar alltid har det stökigt hemma i studentlägenheten.

2.1 Kravspecifikation

Arbetet med att upprätta en kravspecifikation började tidigt och gjordes med stor omsorg för att undvika problem senare vid konstruktion av prototypen. De krav som ställdes på prototypen var följande:

- Little BrushBot ska åka framåt av sig själv utan att vara fast förankrad i en strömkub
- Little BrushBot ska kunna sopa med en borste bakom sig för att göra rent golvet
- Little BrushBot ska kunna upptäcka hinder som är cirka 50 centimeter bort och beroende på situation ska roboten svänga höger eller vänster för att undvika hindret
- Om Little BrushBot fastnar vid ett hinder när de är så nära som 15 centimeter ifrån den och låser sig där, ska roboten backa och manövrera bort
- Dioder ska användas för att öka användarvänligheten - en grön diod ska lysa när roboten är igång och en röd diod ska vara tänd då roboten har identifierat ett hinder. En vit lyser då borsten är aktiv.

2.2 Hårdvara

Little BrushBot består av ett antal olika komponenter som listas nedan. För en mer detaljerad bild av hur de olika komponenterna i prototypen är sammankopplade, refereras till kopplingsschemat som finnes i Appendix 1.

Processor
ATmega 1284. En avancerad 8-bitars processor används för att ha möjlighet att använda fyra stycken olika timers.

Servomotor
Goteck GD-9257. Två stycken servomotorer används i prototypen. Den ena används för att rotera ultraljudssensor med syfte att upptäcka hinder. Den andra används för att rotera borsten.

Lysdioder
Tre lysdioder, en grön, en vit och en röd. Används för att signalera att Little BrushBot är igång och den vita lyser när borsten används. Den röda används för att indikera att det är ett hinder i vägen.

H-Brygga
L298. En dubbel H-brygga används för att kunna ändra riktning på motorerna.

Strömbrytare
En strömbrytare används för att kunna starta och stänga av roboten.

Ultraljudssensor
SHARP 2Y0A02. Används för att få roboten att ändra riktning. Sitter monterad på en servomotor som roterar för att identifiera hinder.

Batteri
Ett 6V-batteri som används för att förse alla komponenter med ström.

Resistorer
Resistorer med olika motstånd används.

JTAG
AVR JTAG. Används för att överföra kod från datorn till processorn.

DC-motor
Två stycken drivenheter används för att kunna förflytta roboten.

2.3 Mjukvara

Den mjukvara som finns till Little BrushBot är kodad med hjälp av programmeringsspråket C och har kodats i Atmel Studio 7.0. Kopplingsschemat konstruerades med hjälp av programmet Eagle. För att få roboten att utföra det som är skrivet i programmet användes en JTAG. Både kopplingsschemat och källkoden återfinns i Appendix A och B.

I stora drag är mjukvaran konstruerad med en enda stor main-klass. Klassen är uppdelad i en main-metod där roboten initieras och därefter en oändlig while-sats som anropar andra metoder beroende på om användaren klickar på knappen och byter mellan att borsten är stilla till att den rör sig. De metoder som finns är bland annat en som kollar om knappen är nedtryckt, en som roterar servot för att styra ultraljudssensorn tillika avståndsmätaren, en som roterar servot för att förflytta borsten och en för att identifiera hinder. Beroende på vart hindret är så kommer olika förflyttningsmetoder att anropas.

3. Genomförande

I följande avsnitt beskrivs hur genomförandet av projektet har gått tillväga.

3.1 Planering och förberedelser

Projektet startade med att gruppen spånade idéer till vad för typ av prototyp som hade varit rolig att göra. Gruppen kom fram till att en självkörande robot skulle vara intressant att bygga. Det första som hände därefter var att en kravspecifikation togs fram för att definiera vad gruppen ville uppnå med roboten. Därefter var det dags att välja vilka komponenter som behövdes för att färdigställa prototypen. Detta gjordes med hjälp av informationen i respektive komponents datablad och med stöd av handledaren. När komponenterna väl var valda, var det dags att konstruera ett kopplingsschema att ha som stöd vid monteringen av prototypen. Kopplingsschemat sågs över av handledaren och efter godkännande, började montering av prototypen.

3.2 Montering av hårdvara

Med hjälp av det framtagna kopplingsschemat började monteringen av komponenterna. Komponenterna löddes fast på ett kretskort. Kopplingen mellan de olika komponenterna gjordes genom att använda koppartråd som virades fast på pinnarna. Vissa kablar, till exempel de till DC-motorerna, var tvungna att lödas ihop eftersom de var så pass tjocka. För att undvika svårlösta kopplingsfel senare i projektet användes en multimeter för att säkerställa att alla komponenter kopplats rätt och att det gick ström genom dem.

3.3 Programmering av mjukvara

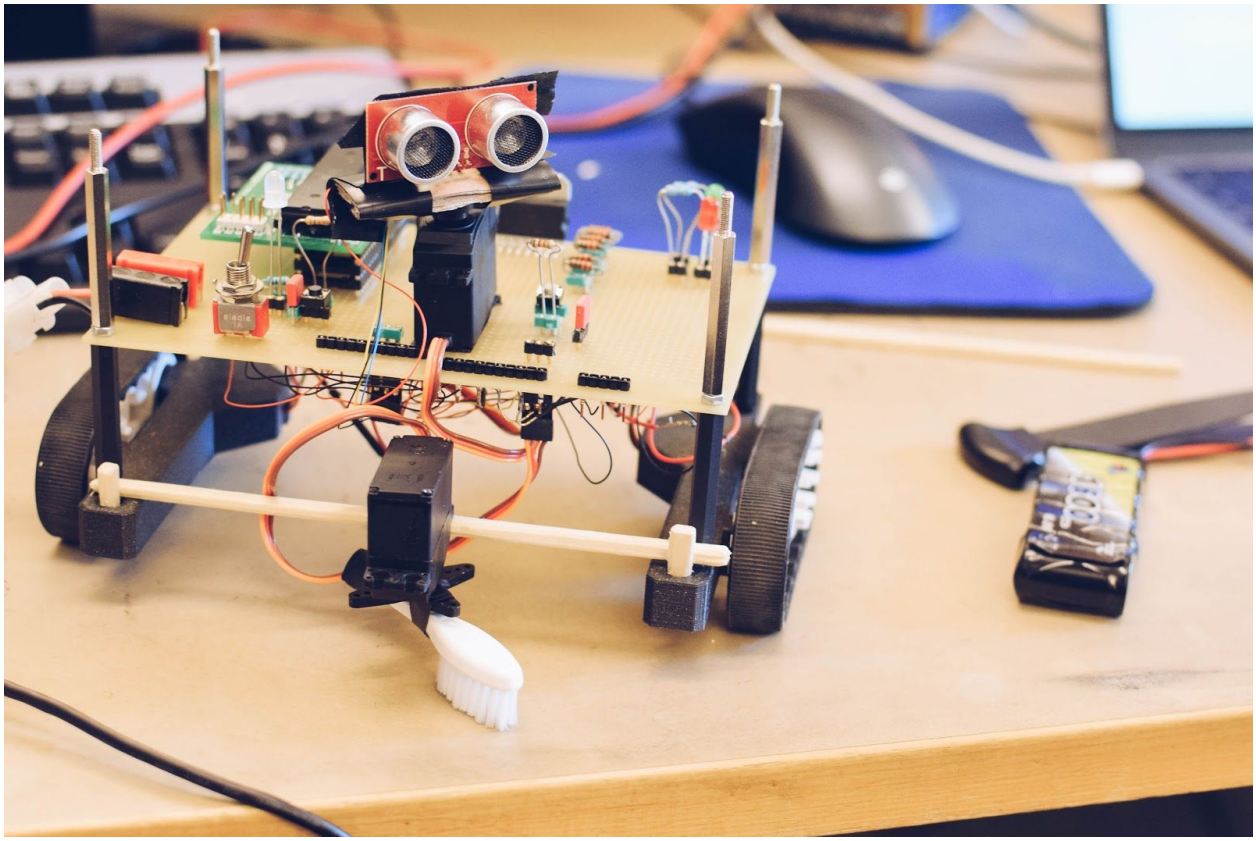
Efter att alla komponenter monterats på kretskortet var det dags att programmera mjukvara till roboten. Programmeringen skedde med hjälp av språket C i Atmel Studio 7.0. Detta var en av de svåraste delarna då ingen av medlemmarna i gruppen hade kodat i andra språk än Java tidigare. Med hjälp av handledaren, kursare och olika webbsidor började mjukvaran att ta form. Det första som gjordes var att skriva kod för att få en av dioderna att lysa. Detta var sannerligen en milstolpe i projektet.

Därefter programmerades mjukvaran till DC-motorerna för att få roboten att förflytta sig. Programmeringen gick framåt och i slutet bestämdes det att lägga till en funktion för att öka komplexiteten. Från början var tanken att bara ett läge skulle finnas, ett automatläge där borsten rör sig men nu används knappen för att få borsten att börja röra på sig. Det ursprungliga läget är med andra ord att roboten åker runt utan att borsta och genom att hålla in knappen så börjar Little BrushBot borsta.

Koden testades kontinuerligt genom att använda JTAGen och undersöka hur roboten rörde sig för att identifiera de buggar som fanns i koden.

4. Resultat

Efter sju kämpiga veckor som kantats av frustration och glädjetårar var prototypen äntligen färdig. Efter en vecka med testkörningar ute i korridoren, korrigerades småfel och gruppen kunde stolta medge att de hade en robot som uppfyller kraven i kravspecifikationen.



5. Användarguide

Nedan följer en manual för att använda Little BrushBot. Generellt sett är den väldigt lättanvänd men för att öka användarvänligheten ville gruppen skriva en användarguide.

1. *Starta Little BrushBot* - Genom att slå på strömbrytaren kommer Little BrushBot att vakna till liv och är redo att köra. Default mode är att roboten kör framåt och undviker föremål som är inom 50 cm framför roboten. Vid längre avstånd än 30 cm gör roboten en ansats att svänga mjukt, vid kortare roterar roboten på stället.

2. *Byt till borstläge* - Håll in knappen i en halv sekund för att byta från vanligt läge till *borstläge*. När Little BrushBot fått order om *borstläge* kommer den att börja borsta med borsten. När den identifierar ett hinder inom 50 cm kommer den att ändra riktning för att undvika hindret, precis som i default läget.

3. *Stäng av Little BrushBot* - När användaren känner sig mätt på fascination av roboten är det bara att slå av strömbrytaren för att få Little BrushBot att stanna och vila.

6. Diskussion

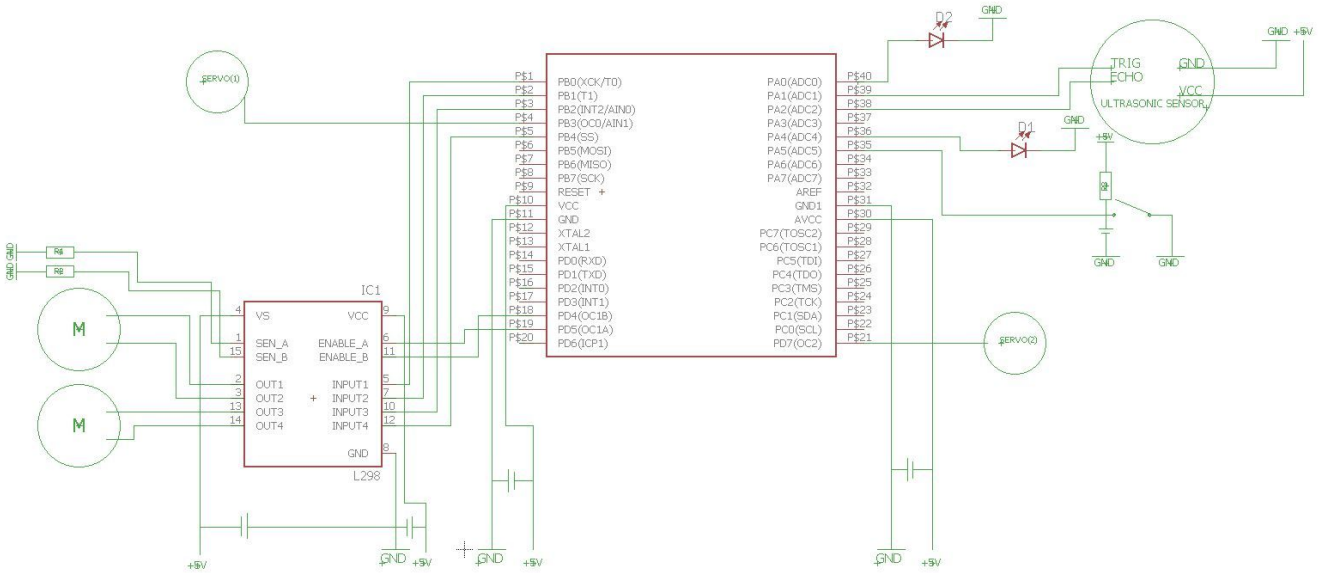
Det här har varit ett av de mest utmanande projekt som gruppen har ställts inför. Det har varit väldigt lärorikt och samtliga gruppmedlemmar känner sig nöjda med vad de har lyckats åstadkomma, men vägen dit har inte varit någon dans på rosor. Det första svåra momentet var att konstruera ett kopplingsschema och ingen av gruppmedlemmarna har någonsin gjort något liknande men tack vare god handledning löste detta sig.

Projektet har överlag flutit på, men den absolut största utmaningen har varit att programmera mjukvaran och att få roboten att bete sig enligt kravspecifikationen. Då inga tidigare erfarenheter av C fanns, var det svårt att sätta fingret på var i koden som programmeringsfelen uppstod.

Det svåraste med projektet har varit att få roboten att undvika föremål på rätt sätt och att försöka förstå hur avståndsmätaren reagerar. Under en tid fungerade roboten som den skulle utom i vissa fall då den började accelerera rakt mot väggen som var bredvid. Ytterligare ett kritiskt moment var när knappen skulle läggas till för att användaren skulle kunna byta läge från automatläge till *borstläge.* På något sätt buggade roboten och slutade fungera helt plötsligt. Detta gjorde att gruppen började ifrågasätta om det verkligen skulle finnas två lägen men till slut lyckades gruppen få Little BrushBot att fungera som önskat.

Eventuella förbättringsåtgärder kunde ha gjorts om det fanns mer tid. Bland annat hade roboten kunnat röra sig ännu mjukare än hur den rör sig nu. Det hade även varit intressant att bygga en radiostyrd robot men det fanns inte tillräckligt med tid.

Appendix 1: Kopplingsschema



Appendix 2: Källkod

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>

#define LEDRED 0
#define REVERSE_R 0 //INPUT1M
#define FORWARD_R 1 //INPUT2M
#define FORWARD_L 2 //INPUT3M
#define REVERSE_L 4 //INPUT4M
#define MOTOR_L 4
#define MOTOR_R 5
#define SERVO1 3
#define SERVO2 7
#define TRIGPORT 1     //port 39
#define ECHOPIN 2      //pin 38
#define BTNPIN 3       //pin 37
#define LEDWHITE 4  //pin 36
#define BTN2PIN 5      //pin 35
#define false 0
#define true 1

static volatile int pulse = 0;
static volatile int i = 0;
volatile uint8_t servo_min = 15;
volatile uint8_t servo_max = 30; //actual max 33
volatile uint8_t t3o_cnt;
double distance = 0;
uint8_t turnDirection = -1;
uint8_t turnPulse1 = 15;
uint8_t turnPulse2 = 15;
uint8_t noBrush = 1;
uint8_t pressed = 0;
uint8_t trapCounter = 0;
uint16_t forward = 0;
uint16_t count = 0;

void initialize(void);
void setup_motor_PWM();
void setup_servo1_PWM();
void setup_servo2_PWM();
void timer0_PWM_output();
void timer1_PWM_output();
void timer2_PWM_output();
void set_PWM_period(uint16_t period);
void set_PWM_motor_pulse(uint16_t pulseA, uint16_t pulseB);
void moveForward();
void moveReverse();
void measureDistance();
void moveStop();
void set_PWM_servo1_pulse(uint16_t pulse);
```

```
void set_PWM_servo2_pulse(uint16_t pulse);
void turnHeadServo();
void checkObstacle();
void rotateLeft(uint16_t delay); //input ms, must be above 255
void rotateRight(uint16_t delay);
void turnRight(uint8_t pulse);
void turnLeft(uint8_t pulse);
void checkButton();
void waveServo();


int main(void){

	initialize();
	setup_motor_PWM();
	setup_servo1_PWM();
	setup_servo2_PWM();
	timer1_PWM_output();
	timer0_PWM_output();
	timer2_PWM_output();
	set_PWM_period(512);
	set_PWM_motor_pulse(300, 300); //speed left, speed right

	sei();

	if(!noBrush){
		PORTA |= (1<<LEDWHITE);
	}
	else{
		PORTA &= ~(1<<LEDWHITE);
	}

    while (1)
    {
		turnHeadServo();
		measureDistance();

		//Calculation   of   distance   =   (count*(prescaler/clockfrequency)*speedOfSound
[cm/s]/2) = ((256/16000000))*34000)/2;
		distance = count*0.272;
		checkButton();
		checkObstacle();

		if(noBrush){
			PORTA &= ~(1<<LEDWHITE);
		}
		else if(!noBrush){
			waveServo();
			PORTA |= (1<<LEDWHITE);
		}

	}
}

void initialize(){
	//Initiate red LED
	DDRA |= (1<<LEDRED);
```

```
	//Initiate green LED
	DDRA |= (1<<LEDWHITE);

	//Initiate ultrasonic sensor
	DDRA |= (1<<TRIGPORT); //start trig-port
	DDRA &= ~(1<<ECHOPIN); //start echo-pin
	TIMSK3 |= (1 << TOIE3);

	//Initiate motors
	DDRB |= (1<<REVERSE_R)|(1<<FORWARD_R)|(1<<FORWARD_L)|(1<<REVERSE_L);

	//Initiate servos, DUPLICATE
	DDRB |= (1<<SERVO1);

	//Initiate buttons
	DDRA &= ~(1<<BTNPIN);
	DDRA &= ~(1<<BTN2PIN);
}


void checkButton(){
	while(!(PINA & (1<<BTN2PIN))){ //wait when button is pressed
		_delay_ms(100);
		if(!(PINA & (1<<BTN2PIN))){
			pressed += 1;
		}
	}

	if(pressed > 5){
		if(noBrush == 1){
			noBrush = 0;
			PORTA &= ~(1<<LEDWHITE);
		}
		else if(noBrush == 0){
			noBrush = 1;
			PORTA |= (1<<LEDWHITE);
		}
	}
	pressed = 0;
}

void turnHeadServo(){
	if(turnPulse2 == servo_max || turnPulse2 == servo_min){
		turnDirection *= -1;
	}

	turnPulse2 += turnDirection;
	set_PWM_servo2_pulse(turnPulse2);
}

void waveServo(){
	if(turnPulse1 == 30){
		turnPulse1 = 17;
		set_PWM_servo1_pulse(17);
	}
	else{
```

```
                turnPulse1 = 30;
                set_PWM_servo1_pulse(30);
        }
}

void checkObstacle(){

        if(distance < 15){
                trapCounter++;
                _delay_ms(20);
        }

        if(distance < 50){
        set_PWM_motor_pulse(300, 300);

        PORTA |= (1<<LEDRED);

                if(turnPulse2 > servo_max - 4 && distance > 30){
                        turnRight(turnPulse2);
                        turnPulse2 = servo_max-3;
                }
                else if(turnPulse2 > servo_max - 4){
                        rotateRight(100*(1+servo_max-turnPulse2));
                        turnPulse2 = servo_max-3;
                }
                else if(turnPulse2 < servo_min + 4 && distance > 30){
                        turnLeft(turnPulse2);
                        turnPulse2 = servo_min+3;
                }
                else if(turnPulse2 < servo_min + 4){
                        rotateLeft(100*(1+turnPulse2-servo_min));
                        turnPulse2 = servo_min+3;
                }

        }
        else{
                PORTA &= ~(1<<LEDRED);
                //Important to reset motor_pulse to avoid lasting effect from turning motion
                set_PWM_motor_pulse(300, 300);
                moveForward();
                trapCounter = 0;
        }

        if(trapCounter > 20){
                moveStop();
                _delay_ms(200);
                moveReverse();
                _delay_ms(1000);
                rotateRight(400);
                trapCounter = 0;
                set_PWM_motor_pulse(300, 300);
                distance = 0; //Reset distance to avoid bugging
        }
}

void rotateRight(uint16_t delay){
        PORTB &= ~(1<<REVERSE_R);
```

```
        PORTB &= ~(1<<FORWARD_L);
        PORTB |= (1<<FORWARD_R);
        PORTB |= (1<<REVERSE_L);
        _delay_ms(delay);
        moveStop();
}

void rotateLeft(uint16_t delay){
        PORTB &= ~(1<<FORWARD_R);
        PORTB &= ~(1<<REVERSE_L);
        PORTB |= (1<<FORWARD_L);
        PORTB |= (1<<REVERSE_R);
        _delay_ms(delay);
        moveStop();
}

void turnRight(uint8_t turnPulse){
        set_PWM_motor_pulse(300-6*(servo_max-turnPulse)*(servo_max-turnPulse), 500);
        PORTB &= ~(1<<REVERSE_R);
        PORTB &= ~(1<<REVERSE_L);
        PORTB |= (1<<FORWARD_L);
        PORTB |= (1<<FORWARD_R);
        _delay_ms(100);
}

void turnLeft(uint8_t turnPulse){
        set_PWM_motor_pulse(500, 300-6*(turnPulse-servo_min)*(turnPulse-servo_min));
        PORTB &= ~(1<<REVERSE_R);
        PORTB &= ~(1<<REVERSE_L);
        PORTB |= (1<<FORWARD_L);
        PORTB |= (1<<FORWARD_R);
        _delay_ms(100);
}

void measureDistance(){

        t3o_cnt = 0;
        PORTA |= (1<<TRIGPORT); //on
        _delay_us(10);
        PORTA &= ~(1<<TRIGPORT);//off

        while(!(PINA & (1<<ECHOPIN))); //while ECHOPIN is low
        TCCR3B |= (1<<CS32);  //start timer
        while(PINA & (1<<ECHOPIN));      //while ECHOPIN is high
        TCCR3B &= ~(1<<CS32); //stop timer

        count = TCNT3;
        _delay_ms(60);//specified according to data sheet
        TCNT3 = 0;
}

void setup_motor_PWM(){
        //Setting timer properties for timer 1, for the motors. 16-bit timer.
        TCCR1A |= (1<<COM1A1)|(1<<COM1B1)|(1<<WGM11); //COM1n1: Clear Compare Match (shall trigger
pulse at certain clock time), WGMn1: Fast PWM (mode)
        TCCR1B |= (1<<WGM13)|(1<<WGM12)|(1<<CS12);   //WGMn2:  Fast  PWM,  CSn2:  Pre-scaler  256
(choosen according to requirements for pulse speed)
```

```
}

void setup_servo1_PWM(){
	//Setting timer properties for timer 0, for the servo. 8-bits timer (no ICR to be set, always 255).
	TCCR0A |= (1<<COM0A1)|(1<<WGM01)|(1<<WGM00); //COM1n1: Clear Compare Match (shall trigger pulse at certain clock time), WGMn1: Fast PWM (mode)
	TCCR0B |= (1<<CS02)|(1<<CS00);//Apparently no WGMn2: CSn2: pre-scaler 1024 (choosen according to requirements for pulse speed)
}

void setup_servo2_PWM(){
	//Setting timer properties for timer 2, for the servo. 8-bits timer (no ICR to be set, always 255).
	TCCR2A |= (1<<COM2A1)|(1<<WGM21)|(1<<WGM20); //COM1n1: Clear Compare Match (shall trigger pulse at certain clock time), WGMn1: Fast PWM (mode)
	TCCR2B |= (1<<CS22)|(1<<CS21)|(1<<CS20);     //CSn2: pre-scaler 1024 (choosen according to requirements for pulse speed)
}

void timer2_PWM_output(){
	//Start servo 2 port
	DDRD |= (1<<SERVO2);

}

void timer1_PWM_output(){
	//Start enableA and enableB
	DDRD |= (1<<MOTOR_L)|(1<<MOTOR_R);

}

void timer0_PWM_output(){
	//Start servo 1 port
	DDRB |= (1<<SERVO1);

}

void set_PWM_period(uint16_t period){
	//Set period i.e. on-time, set to 512
	ICR1 = period;
}


void set_PWM_motor_pulse(uint16_t pulseLEFT, uint16_t pulseRIGHT){
	//Set speed of motor between ~100-500
	OCR1A = pulseRIGHT;
	OCR1B = pulseLEFT;
}

void set_PWM_servo1_pulse(uint16_t pulse){
	//Set angle of servo
	OCR0A = pulse;
}

void set_PWM_servo2_pulse(uint16_t pulse){
```

```
        //Set angle of servo
        OCR2A = pulse;
}

void moveForward(){
        PORTB |= (1<<FORWARD_R);
        PORTB |= (1<<FORWARD_L);
        PORTB &= ~(1<<REVERSE_R);
        PORTB &= ~(1<<REVERSE_L);
}

void moveReverse(){
        PORTB &= ~(1<<FORWARD_R);
        PORTB &= ~(1<<FORWARD_L);
        PORTB |= (1<<REVERSE_R);
        PORTB |= (1<<REVERSE_L);
}

void moveStop(){
        PORTB &= ~(1<<FORWARD_R);
        PORTB &= ~(1<<FORWARD_L);
        PORTB &= ~(1<<REVERSE_R);
        PORTB &= ~(1<<REVERSE_L);
}

ISR(TIMER3_OVF_vect) {

        t3o_cnt++;

}
```