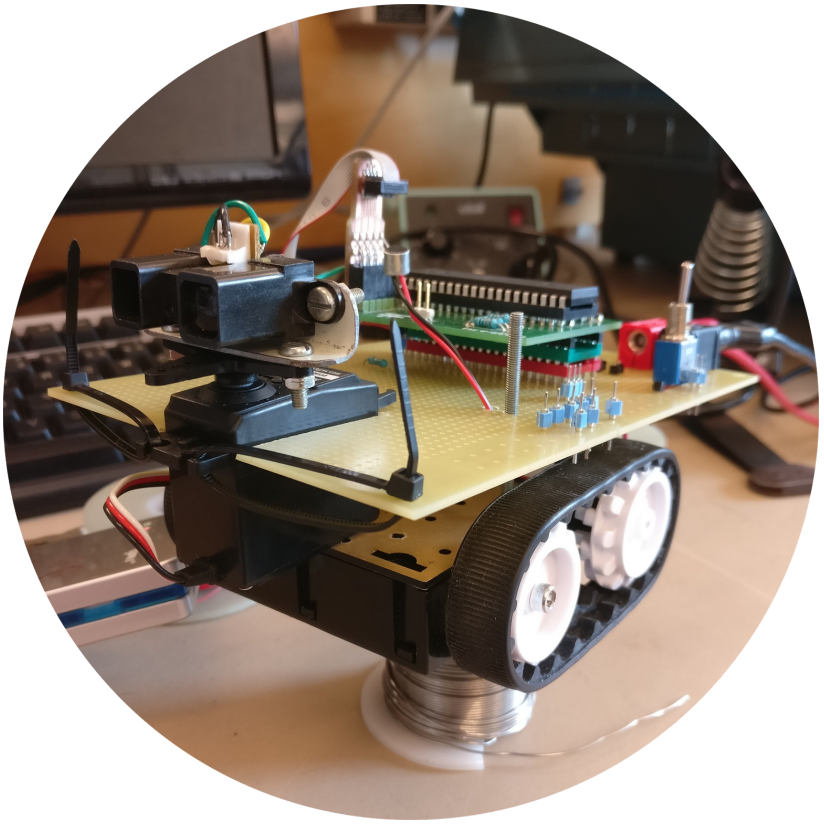
Institutionen för elektro- och informationsteknik vid LTH



Från idé till skapelse av en äkta RoomBot

Projektarbete i kursen EITF11 - Digitala Projekt

Gustav Handmark, Claudio Gandra & Kasper Trolltoft - I15

2018-05-21

Abstract

The purpose of this project was to build a working prototype by constructing hardware and developing appropriate software. The chosen type of prototype, the RoomBot, is a robot similar to a robot vacuum cleaner, excluding the vacuuming feature. The main requirements are that RoomBot is supposed to being able to drive forwards, turning and reversing while being able to detect and steer clear of objects in front of itself.

To construct such a prototype the project group thought of what components that would be needed, drew a wiring diagram, put all the components together and developed all the necessary code to handle each feature and component. After many hours, a few hiccups and lots of new knowledge acquired, the group was finally able to produce a working, and in their mind impressive, prototype.

Innehållsförteckning

1. Inledning

I kursen EITF11 - Digitala projekt skall studenterna i grupp om 2-3 personer konstruera en fungerande prototyp med tillhörande dokumentation. Detta sker genom framtagning av ett kopplingsschema, koppling av komponenter samt utveckling av tillhörande mjukvara. Vilken typ av prototyp som ska tas fram och exakta krav på denna väljs av studenterna i samråd med handledare.

I en tid där artificiell intelligens och robotar är hett omdiskuterade ämnen valde denna grupp att konstruera en robot liknande en robotdammsugare. Roboten kan utan mänsklig input köra runt och undvika att köra in i hinder, samt reagera på ljud i omgivningen. Syftet med denna rapport är att beskriva utvecklingsprocessen och den färdiga produktens ingående komponenter och funktioner. I kravspecifikationen beskrivs de krav som ställs på roboten och hur den ska agera i olika situationer. I slutet utvärderas projektet och det slutliga resultatet.

2. Kravspecifikation

Följande krav ställs på den färdiga prototypen:

1. Roombot ska upptäcka objekt 15 cm framför den, och då backa en liten bit för att sedan svänga åt ett visst håll beroende på var det upptäckta objektet befinner sig. Därefter ska RoomBot fortsätta framåt i den nya riktningen tills dess att den stöter på ett nytt objekt.
2. RoomBot ska kunna röra sig fritt i rummet utan anslutning till ett fast objekt.
3. RoomBot ska ha en strömbrytare där strömmen ska kunna slås på och av.
4. RoomBot ska ha en grön och en röd lysdiod. Den gröna ska vara tänd då strömmen är på, annars släckt. Den röda ska tändas då RoomBot upptäcker ett objekt framför sig.
5. RoomBot ska upptäcka ljud i dess närhet. Vid en första klapp med händerna ska RoomBot starta, och vid en andra klapp ska RoomBot stanna.

3. Teori

3.1 Hårdvara

Nedan följer en beskrivning av de komponenter RoomBot är uppbyggd av. Hur dessa komponenter är kopplade till varandra kan utläsas från kopplingsschemat som hittas i appendix A.

Processor
ATmega16, 8-bitars processor. Denna programmeras för att styra övriga komponenter enligt önskat beteende.

Servomotor
Parallax Standard Servo. Används för att rotera avståndssensorn medan RoomBot kör, för att kunna upptäcka objekt i ett större område.

Avståndsmätare
Sharp GP2Y0A02YK. Används för att upptäcka objekt framför RoomBot och är monterad ovanpå servomotorn. Sensorn avger en spänning med en viss styrka beroende på hur långt ifrån objektet befinner sig, och vid en viss spänning reagerar RoomBot.

Lysdioder
Två lysdioder, en röd och en grön. Den gröna lyser när strömmen är igång och den röda när RoomBot upptäcker ett objekt.

H-brygga
DRV8833. Används för att styra strömmen till DC-motorerna.

Drivenhet
Lawicel Zumo Chassis Kit med två DC-motorer. Används för att driva RoomBots förflyttning i rummet.

Strömbrytare
En standard strömbrytare för att slå på och av strömmen till RoomBot.

Resistorer
Tre vanliga (1x 130 Ω, 2x 190 Ω) och en modulär resistor. Används för att justera spänningen till olika komponenter.

Batteri
Ett batteri med spänningen 6 volt. Används för att förse samtliga komponenter med ström.

JTAG
AVR JTAG. Används för att skicka in kod från datorn till processorn.

3.2 Mjukvara

Koden till RoomBot är skriven i språket C och utvecklad i Atmel Studio 7. Källkoden finns i appendix B.

Robotens kod exekveras kontinuerligt när strömbrytaren slås på, huvudsakligen består koden av en main metod som i sin tur kallar metoder för att initiera variabler, ADC-omvandlaren, PWM frekvenser för servomotorn och insignaler till H-bryggan. När detta är färdigt fortsätter koden i en oändlig while-loop som först kollar huruvida mikrofonen har fått en tillräcklig hög input för att starta bilen. När mikrofonen får en sådan signal startas logiken för att styra bilen, denna fungerar genom att kalla metoderna drive och Obstacle. I ursprungsläge skickas kontinuerliga signaler till H-bryggan som vidarebefordrar dessa till de två DC-motorerna i chassiet för att köra rakt fram.

Servomotorn drivs kontinuerligt med hjälp av globala avbrott som roterar motorn och således även sensorn som är monterad ovanpå. ADC omvandlingen för att upptäcka hinder fungerar även denna med hjälp av avbrott som kör kod för att medelvärdesbilda resultatet från sensorn. Vid ett tröskelvärde, som uppstår när ett hinder cirka 15 till 30 cm från bilen påträffas, körs kod för att undvika hindret. När sensorn upptäcker ett hinder i någon riktning ändras binära variabler för att avgöra i vilken riktning bilen ska fortsätta, detta genom att använda ytterligare hjälpmetoder som kontrollerar huruvida en motor ska stanna eller drivas medsols eller motsols. Vid ett hinder drivs alltså den ena motorn motsols, och den andra motorn medsols (i bilens färdriktning) för att rotera chassiet och undvika hindret. Uppstår en situation där alla riktningar är blockerade (exempelvis i ett hörn) försöker bilen backa ut från hörnet, för att sedan fortsätta igen. Dessvärre är sensorn begränsad på så sätt att den ej kan fånga upp föremål som är närmare än 15 cm, vilket gör att bilen kan fastna.

4. Genomförande

4.1 Planering

Då den ursprungliga idén endast bestod i att någon typ av robot skulle konstrueras behövde denna robots funktioner specificeras ytterligare. Det första steget bestod alltså i att en kravspecifikation togs fram. Efter att kravspecifikationen tagits fram identifierades vilka komponenter som skulle behövas för att uppfylla kraven. För att få information om komponenternas funktioner och specifikationer studerades respektive datablad. Utifrån informationen i databladen togs ett kopplingsschema fram, vilket slutligen godkändes av handledaren.

4.2 Konstruktion av hårdvara

Efter att kopplingsschemat blivit godkänt av handledaren påbörjades konstruktionen. Efter övning i lödning och introduktion till koppling kopplades komponenterna samman enligt kopplingsschemat. Samtliga komponenter utom chassi och avståndssensor med servomotor

löddes fast på ett mönsterkort. Kopplingen mellan komponenterna skedde med koppartråd som virades och löddes fast. För att fästa mönsterkortet med alla komponenter på chassit gjordes två hål i mönsterkortet som sedan lades över två från chassit uppstickande metallstavar. För servomotorn med avståndssensor skars en passande rektangel ut i mönsterkortet i vilken motor med sensor placerade. Samtliga kopplingar testades sedan med en multimeter för att försäkra att komponenterna mottog och sände ut ström korrekt.

4.3 Programmering av mjukvara

Med samtliga kopplingar och komponenter fungerande som de skulle övergick arbetet till att koda den nödvändiga mjukvaran i Atmel Studio 7. Då gruppmedlemmarna endast kodat Java innan behövde kunskap om C inhämtas, detta gjordes främst genom att läsa på på olika webbsidor. Efter att ha lärt sig grunderna påbörjades kodning relaterad till en komponent i taget. Först säkerställdes att lysdioderna fungerade som önskat, sedan servomotor, avståndssensor, drivlinan med de två DC-motorerna och till sist mikrofon.
Det visade sig snart att mikrofonen som monterades på i slutet inte fungerade som tänkt. Efter att problemet var identifierat fick gruppen nöja sig med att den simpla klapp som var tänkt för att kommunicera med roboten med behöver vara oerhört nära och högljudd, eller så fungerar det bättre att blåsa i mikrofonen.

5. Resultat

Resultatet av projektet blev en livs levande RoomBot! Med detta menas en robot som uppfyller samtliga krav i kravspecifikationen (om än att handklapp är opålitligt). Användarmanual för den färdiga prototypen hittas i nästa avsnitt. För beskrivande bilder och video, se hemsidan.

6. Användarmanual

Då tanken med RoomBot är att den ska vara autonom är mängden användarfall starkt begränsade. Nedan följer en förklaring på hur RoomBot används.

1. Sätt på strömmen
För att sätta på strömmen till RoomBot trycks spaken på strömbrytaren så den riktas akterut.

2. Starta/stoppa RoomBot
För att RoomBot ska börja köra klappar användaren en gång högljutt med händerna i närheten av RoomBot eller blåser i mikrofonen (rekommenderat). För att RoomBot ska stanna klappar användaren med händerna/blåser en gång till på samma sätt.

3. Slå av strömmen
För att slå av strömmen till RoomBot trycks spaken på strömbrytaren så den riktas framåt.

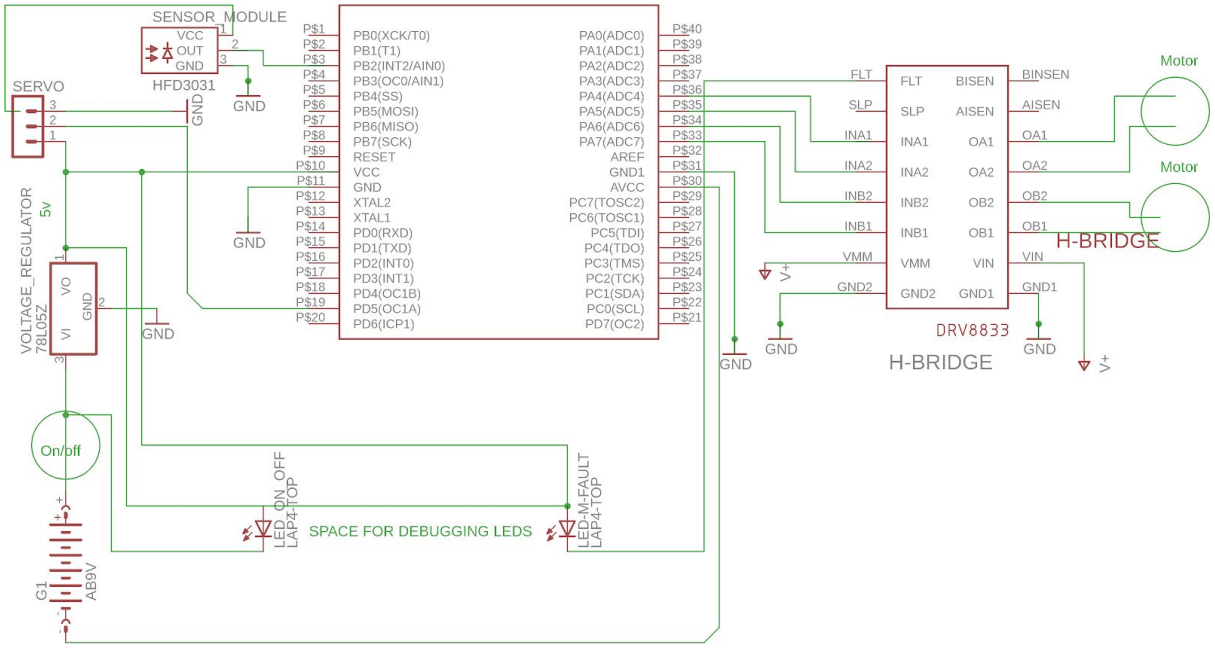
7. Diskussion

Projektet har varit stundtals utmanande men oerhört intressant och lärorikt. Många av momenten var nya för gruppmedlemmarna, och arbetet med hårdvara och koppling av komponenter var annorlunda mot tidigare kurser för oss som I-studenter. Den del av projektet som var överlägset mest tidskrävande och utmanande var dock kodningen av roboten. För det första behövde gruppen lära sig ett helt nytt programspråk, C, då medlemmarna endast kodat Java tidigare, och för det andra var det många funktioner vi inte stött på tidigare. Att förstå och koda delen med Pulse Width Modulation presenterade den största utmaningen.

Förutom stundtals svår kodning och problemet med mikrofonen som nämns under avsnittet “genomförande” gick projektet relativt smidigt. Målsättningen med prototypen var ambitiös, men efter att ha lagt ner mycket tid och energi är gruppen mycket nöjda och imponerade över resultatet. Vid fortsatt arbete hade prototypen dock självklart kunnat utvecklas ytterligare. En relativt enkel sak hade varit att byta ut den nuvarande mikrofonen mot en med bättre prestanda, förslagsvis en med inbyggd förstärkare. Detta hade gjort att RoomBot lättare hade kunnat uppfatta ljud som klappar med mera. Även RoomBots avståndssensor hade kunnat bytas ut mot en som kunde upptäcka saker närmare än 15 cm, vilket hade eliminerat risken för att RoomBot fastnar. Ytterligare en förbättring man hade kunnat göra skulle vara att utveckla algoritmen för RoomBots navigation. Dagens navigation fungerar bra i de flesta fall, men genom att ta mer hänsyn till vinklar till det upptäckta objektet etc. hade man kunnat göra den ännu mer träffsäker.

Appendix

A. Kopplingsschema



B. Källkod

```
/*
 * GccApplication1.c
 *
 * Created: 4/18/2018 12:21:35 AM
 * Author : Gustav
 */

#define _BV(bit) (1<<(bit))
#define clear_bit(a,z) (a &= ~_BV(z))
#define set_bit(a,z) (a|=_BV(z))
#define ADCH0 ((1<<(REFS0)) | 0)
#define ADCH1 ((1<<(REFS0)) | 1)
#define ADCH2 ((1<<(REFS0)) | 2)
#define ADCH3 ((1<<(REFS0)) | 3)
#define F_CPU 8000000UL                 /* Define CPU Frequency  */
#include <avr/io.h>                     /* Include AVR std. library file */
#include <stdio.h>                      /* Include std. library file */
#include <util/delay.h>                         /* Include Delay header file */
#include <avr/interrupt.h>
typedef int bool;
```

```
#define true 1
#define false 0
uint8_t stopDist, dist1, dist2, leftDist, rightDist;
uint16_t adc_result, adc_sum,angle,angle2, adc_mic_sum;
uint8_t adc_count, adc_mic_count;
volatile int analogV1;
volatile int analogV2;
volatile int analogV3;
volatile int adcCh;
int dist;
bool nextDist, adc_mean_done, adc_done, clap_done, right_block, left_block, center_block,
on, off;

//DC Motors

void initMotor1(){
	DDRD |= _BV(DDD7); //set PD7 OC2 to output)
	DDRB |= _BV(DDB5); //() (PB5) for motor ctrl set PB5 output
	TCCR2 |= _BV(WGM20); //set Phase correct PWM mode.
	TCCR2 |= _BV(COM21); //clear on compare match, non-inverted.
	TCCR2 |= _BV(CS21); //prescale divider 1.

}

void forwardM1(){
	set_bit(PORTB,PB5);
	OCR2 = 185; //170
}
void reverseM1(){
	clear_bit(PORTB,PB5);
	OCR2 = 80; // 85
}
void breakM1(){
	set_bit(PORTB,PB5);
	OCR2 = 255;
}

//Motor two. (Hbridge BIN1)s
void initMotor2(){
	DDRB |= _BV(DDB3); // set PB3 OC0 as output.
	DDRB |= _BV(DDB4); // set PB4 as output. (PA1 to bin2) for motor ctrl
	TCCR0 |= _BV(WGM00); //set Phase correct PWM Mode.
	TCCR0 |= _BV(COM01); // clear on compare match. non-inverted.
	TCCR0 |= _BV(CS01);// prescaler divider 1.
}
```

```
void forwardM2(){
	set_bit(PORTB,PB4);
	OCR0 = 190; //170
}
void reverseM2(){
	clear_bit(PORTB,PB4);
	OCR0 = 80; //85
}
void breakM2(){
	set_bit(PORTB,PB4);
	OCR0 = 255;
}


void initVars(){
	on = false;
	off = true;

}

//ServoController
void initPWMServo()
{
	DDRD |= _BV(DDD4); // OC1B as output

	TCCR1A |= _BV(COM1B1);  // Clear OC1A/OC1B on compare match

	TIMSK |= _BV(TOIE1);
	TCNT1H = 0x00;
	TCNT1L = 0x00;

	ICR1 = 10000; //50Hz
	TCCR1B |= _BV(WGM13) // mode 8, PWM, Phase and Frequency Correct (TOP
value is ICR1)
	|  _BV(CS11); //prescaler 8
}
/**
	Sets servo position.
	\param Angle in degrees from 0 to 179

	Parallax Standard Servo, PWM from 0.75 ms - 2.25 ms, where 1.5 ms is center.
	Pulses: for 20 ms intervals, use:
	10000/20ms => OCR = 500/ms.
	so: turning servo from min position to max position (0 - 180 degrees)

	MinOCR = 0.75*500 = 375
```

```
        MaxOCR = 2.25*500 = 1125 (but, 10 bit resolution => max 1023.


*/
void SetServoPosition(int a)
{
        const uint16_t MinOCR = 400;

        const uint16_t MaxOCR = 770;

        uint16_t ocr =  MinOCR + ((MaxOCR-MinOCR) * ((a*100)/180))/100;

        OCR1B = ocr;
}

void initADC(){
        DDRA = 0x00;
        ADMUX = ADCH0;
        ADCSRA |= _BV(ADPS2)|_BV(ADPS1)|_BV(ADPS0)|_BV(ADIE)|_BV(ADEN);
        sei();

        analogV1=0;
        analogV2=0;
        ADCSRA |= (1<<ADSC);


}


void Obstacle(){
        if(adc_mean_done){
                adc_mean_done = false;
                if(analogV1 > 450) {
                        if(OCR1B < 440) {
                                right_block = true;
                        }

                        if(OCR1B > 730) {
                                left_block = true;
                        }

                        if(OCR1B > 560 && OCR1B < 590) {
                                center_block = true;
                        }
                }
                else {
```

```
                forwardM1();
                forwardM2();
            }

        }
}

void drive() {
            if(right_block && left_block && center_block) {
                reverseM1();
                breakM2();
                _delay_ms(1000);
                center_block = false;
                right_block = false;
                left_block = false;
            }

            if(right_block) {
                reverseM2();
                _delay_ms(350);
                right_block = false;
            }

            if(left_block) {
                reverseM1();
                _delay_ms(350);
                left_block = false;
            }
        }


void checkObstacle(){
        if(clap_done){
            clap_done = false;
            if(analogV2 > 45) {
                if(on==false) {
                    on = true;
                    DDRB |= _BV(DDB0);
                    set_bit(PORTB,PB0);
                }
                }
            }
        }


ISR(TIMER1_OVF_vect) {
```

```
        if(angle < 180) {
        SetServoPosition(angle);
        angle=angle + 10;
        }
        if(angle >= 180) {
                SetServoPosition(angle2);
                angle2=angle2-10;
        }
        if(angle2 <=0) {
                angle = 0;
                angle2 = 180;
        }
}


ISR(ADC_vect)
{
        adcCh = ADMUX & (_BV(MUX3) | _BV(MUX2) | _BV(MUX1) | _BV(MUX0));
        uint16_t adc_result = ADC;
        switch(adcCh) {
                case 0:
                analogV1 = adc_result;
                adc_sum = adc_sum + adc_result;

                adc_count++;
                if(adc_count == 8){
                        adc_mean_done = true;
                        analogV1 = adc_sum >> 3;
                        adc_sum   = 0;
                        adc_count = 0;
                        ADMUX = ADCH2;
                }
                break;

                case 2:

                analogV2 = adc_result;
                adc_mic_sum = adc_mic_sum + adc_result;
                adc_mic_count++;
                if(adc_mic_count == 8) {
                        clap_done = true;
                        analogV2 = adc_mic_sum >>3;
                        adc_mic_sum = 0;
                        adc_mic_count = 0;
                        ADMUX = ADCH0;
                }
```

```
                break;


        }
        ADCSRA |= _BV(ADSC);

}


int main()
{
        initVars();
        initADC();
        initPWMServo();
        initMotor1();
        initMotor2();
        while(1){
                checkObstacle();
                if(on) {
                Obstacle();
                drive();
                }
        }
}
```