Självkörande bil

Ellen Niklasson, Alexander Sea-Cho Ek, Lucas Månsson,

Sonny Olsson, Arvid Rosenström Bergkvist

LUNDS UNIVERSITET

Digitala system

2023-05-18

EITA15

ABSTRACT

In this report a prototype of a self-driving car is presented. The purpose with the project was to develop and present their knowledge within digital systems. During the examination of the projekt, work was done with both the hardware and software. As a start a requirement specification was made to clarify the goal with the project. With the clear goal the hardware components were chosen to manage the tasks. Reachers were then done on the various parts in order to create a wire diagram. During the programming, Atmel Studio 7 was used, which is written in the language C. Each component was coded and tested individually until it worked as the desired result. PWM was used to control the intensity of the components.

The project resulted in a self-driving car with the ability to avoid and veer away from objects in its path. Due to the sensor's range to detect objects, the car can turn and deviate from the obstacle without coming to a stop. The projekt has developed and presented the student’s knowledge.

KEYWORDS

ATMEGA 1284, Autonomic car, Hardware Programming, Digital systems

SAMMANFATTNING

Denna rapport presenterar en prototyp av en självkörande bil. Syftet med projektet var att utveckla och redovisa sina kunskaper inom digitala system. Under utförandet av projektet arbetades det med både hårdvaran och mjukvaran. Som en start av projektet skapades en kravspecifikation och planering av passande komponenter för att nå målet. Efterforskning gjordes om de olika delarna för att kunna skapa ett kopplingsschema.

Programmeringen skrevs med Atmel Studio 7 som är skrivet i programmeringsspråket C. Varje komponent kodades och testades individuellt tills den fungerade efter önskat resultat. PWM användes för att styra intensiteten för samtliga komponenter.

Projektet resulterade i en självkörande bil med förmågan att undvika föremål i dess väg. Med hjälp av sensorns räckvidd för att upptäcka föremål, kan bilen svänga och undvika hindret utan att stanna. Projektet har utvecklat och redovisat studenternas kunskap.

NYCKELORD

ATMEGA 1284, Självkörande bil, Hårdvaru Programmering, Digitala system

INNEHÅLLSFÖRTECKNING

1. INTRODUKTION

En av de mest spännande innovationerna inom fordonsindustrin i modern tid är den självkörande bilen. Med ny avancerad teknik har självkörande bilar potentialen till att revolutionera framtidens transport. Förhoppningen är att minska trafikstörningar, öka trafiksäkerheten och förbättra bekvämligheten för passageraren men även föraren. Detta är inte en ny ide med robotar som hjälper oss med vardagen utan det är en fantasi samhället länge har haft, men vi har aldrig varit närmre än idag.

Globalt körs ungefär 16 biljoner kilometer av självkörande bilar varje år, men än idag finns det många utmaningar kvar att lösa.[1]

I detta projektet skulle en prototyp av en självkörande bil i enklare format skapas. Bilen skulle klara de enkla funktionerna som att köra fram, backa och svänga. Även de algoritmer och sensorer som används för att identifiera hinder på vägen och navigera skulle konstrueras. Under utförandet fick projektgruppen arbeta med både hårdvara och mjukvara. Programmeringen skedde med C. Genomförandet av projektet skulle utveckla och fördjupa projektgruppens kunskaper inom projekthantering och projektutveckling

[1] https://ieeexplore.ieee.org/abstract/document/8220479

2. TEKNISK BAKGRUND

I detta avsnitt presenteras de tekniska delarna för projektet. Detta avsnitt är uppdelat i två delar där den första delen handlar om projektets hårdvara och i den andra delen beskrivs mjukvaran.

2.1 HÅRDVARA

Här redogörs de hårdvaror som har använts i projektet samt en beskrivning av dessa komponenters funktioner.

Hårdvaror som har använts i projektet:

- **ATMEGA1284** - Mikrokontroller med 8-bit AVR som styr komponenterna i projektet
- **SHARP 2Y0A02** - infraröd sensor som upptäcker hinder framför bilen
- **Feetech FS90 micro servo** - Servomotor som roterar sensorn fram och tillbaka för att upptäcka hinder runt om bilen
- **DC-motor** - elektrisk motor som omvandlar elektrisk energi till mekanisk energi för att få hjulen att rulla
- **DRV8833** - Integrerad krets som styr hastigheten och riktningen för DC-motorerna
- **Atmel-ICE JTAG** - Debugger som används för att felsöka och kommunicera med mikrokontrollern

Se figur 1 för en utförlig beskrivning av hur komponenterna är integrerade.

2.2 MJUKVARA

I detta avsnitt presenteras vilka verktyg som har använts för programmeringen samt en kort beskrivning av koden.

För att programmera Mikrokontrollern så användes programmeringsverktyget Atmel Studio 7. Programmet är skrivet i programmeringsspråket C och koden debuggades med Atmel ICE på JTAG-gränssnitt.

För att styra hastigheten på hjulen genererades en fyrkantsvåg med pulsbreddsmodulering (PWM), vars pulsbredd bestämmer den hastighet som hjulen snurrar med. Denna PWM - signal skickades till H - bryggan som sedan omvandlade den till en spänning, vilket i sin tur skickades till hjulens ingång. Hjulen hade varsin PWM - signal vilket tillät hastigheten på respektive hjul att ändras individuellt. En PWM - signal användes också för att styra riktningen på den servon som sensorn satt på.

Då sensorn läste av ett hinder skickade den en analog signal med en spänning som beror på avståndet till objektet, upp till 150 cm. En högre spänning innebar att hindret var närmre och en längre att det var längre bort. För att kunna avläsa avståndet behövde spänningen konverteras till ett digitalt värde, vilket gjordes med hjälp av en analog/digital omvandlare (ADC).

2.3 PULSBREDDSMODULERING

För att styra både motorn och servon i den självkörande bilen så användes tekniken pulsbreddsmodulering, även kallad PWM (Pulse Width Modulation). PWM är en teknik som skapar en variabel effektmatning genom att spänningen slås på och av så snabbt att de anslutna komponenterna inte märker av dem. Genom att ändra pulsbredden kan så kan intensiteten eller effekten styras. Om pulsbredden är kortare så blir den genomsnittliga effekten lägre, medan längre pulsbredd skapar en högre genomsnittlig effekt.

PWM används i projektet för att styra hastigheten på hur snabbt servon ska rotera och hur snabbt bilen ska köra.

3. METOD

Projektets metod består av tre delar: planering, hårdvara, och mjukvara. I planeringen beskrivs uppdelningen av arbetsuppgifter och hur samspelet mellan alla medlemmar gick till. I hårdvara förklaras hur gruppen gick till hand att välja samt använda de olika komponenterna. I mjukvara beskrivs metodiken bakom koden som får självkörande bilen att rulla.

3.1 PLANERING

Det första som gjordes i gruppen var att bestämma vad vi skulle bygga för projekt och gruppen kom tidigt överens om att bygga en självkörande bil. Därefter gjordes en kravspecifikation, se tabell 1, och planering för att tydliggöra för alla gruppmedlemmar tillvägagångssättet för att uppnå det önskade resultatet. När planeringen var klar utsågs varje medlem till ansvarig inom ett varsitt område där den ansvariga såg till att det sitt område följde planeringen. De 5 ansvarsområden som fanns var: Hemsida, Rapporten, Kopplingsschema, Hårdvaror, Mjukvara. Därefter gjordes en Discord chat där gruppens medlemmar kunde planera när olika möten skulle hållas samt ifall någon behövde hjälp. Även en Github gjordes där all färdig kod las in.

3.2 HÅRDVARA

Början av projektet var det krav av handledare att lämna in ett kopplingsschema samt en produktlista av komponenter som behövdes för att realisera gruppens idé för att sedan få ut komponenterna i hand. Efter diskussion med handledare uppdaterades den ursprungliga produktlistan till den nuvarande. Handledaren tyckte det inte var lämpligt att använda en ultraljudssensor för den idé gruppen ville realisera men rekommenderade den IR-sensor som används i slutprodukten.

Komponenternas koppling till processorn är baserat på dess pins egenskaper samt hur komponenten förväntar sig att en signal ska se ut. DRV8833 och FS90 tar emot PWM signaler vilket bara pins med OCXX funktion kan skapa. Av denna anledning är DRV8833 som kräver totalt fyra PWM signaler för att styra de två servomotorerna som kör hjulen kopplad till PB3, PB4, PD6 och PD7, samt är IR-sensorns servomotor kopplad till PB6. Själva signalen är sedan skapad i programmeringen av processorn. SHARP 2Y0A02 ger ett analogt värde som måste omvandlas till digitalt innan det kan utnyttjas av processorn. Det är anledningen för kopplingen till PA0 som har en analog to digital conversion funktion. Med hjälp av programmering samt PA0s egenskaper gör mätvärdet som sensorn registrerat läsbart och användbart av

processorn.

3.3 MJUKVARA

Ursprungligen skrevs koden som gjorde att servon och hjulen kunde styras, samt att ett värde kunde avläsas från sensorn. Förklaring för hur dessa komponenter styrdes återfinns på “2.2 Mjukvara” under teknisk bakgrund.

När bilen startar roteras servon ungefär 30° till höger från mittpunkten. Därefter roterar den vänster i steg om 15° ner till ungefär -30° och sedan tillbaka igen, för att sedan upprepa detta kontinuerligt. Sensorns värde uppdateras konstant under förflyttningarna. Då sensorn returnerar ett värde som motsvarar att ett hinder är i vägen, registreras detta och om hindret fortfarande är kvar efter 10 successiva avläsningar (för att undvika felläsningar) kommer hjulens hastighet förändras så att den svänger åt motsatt håll för att undvika hindret. Bilen svänger också olika mycket beroende på i vilken vinkel sensorn är riktad då hindret upptäcks. Då ett hinder upptäcks i spannet 0° till 15° respektive -15° till 0° kommer bilen svänga mer än om den upptäcker något i spannet 15° till 30° respektive -30° till -15°. Efter hindret upptäckts och bilen börjat svänga kommer sensorn stanna kvar i samma position tills hindret är borta för att säkerställa att bilen svängt tillräckligt.

För en mer omfattande överblick av koden, se källkoden under 8.2 C - kod för projektet.

4. ANALYS

	KRAVSPECIFIKATION
1	Skapa en självkörande bil
2	Bilen ska ha förmåga att köra rakt fram
3	Bilen ska ha förmåga att smidigt svänga utan att stanna
~~4~~	~~Bilen ska kunna backa~~
5	Bilen ska kunna undvika att köra in i väggar
6	Bilen ska kunna undvika att fastna i en återvändsgränd
~~7~~	~~Bilen ska kunna undvika att köra nerför ett stup~~

Vid planeringen av projektet kom vi fram till att använda oss av ultraljussencorer. Genom valet av sensor skulle det kunna upptäckas om bilen var på väg att köra ner för en kant. Däremot så ändrades valet av sensor till en infraröd sensor, detta medförde att den kunde röra sig ungefär 50° vägrätt men dessvärre inte lodrätt. Det kan konstateras att krav nummer 7 inte har uppfyllts. För att säkerställa en smidigare funktion för återstående mål beslutades krav nummer 7 att tas bort under projektets gång.

Även krav 4 beslöts att tas bort under utvecklingen av bilen. Den huvudsakliga anledningen till tanken om att backa kom från att inte fastna i en återvändsgräns. Däremot klarar bilen att ta sig ut genom att rotera 180°. Det beslutades då att krav nummer 4 ej behövdes och valdes därför att tas bort för effektivisering av projektet. De återstående målen i kravspecifikationen har realiserats.

5. RESULTAT

Projektet resulterade i en självkörande bil som med hjälp av sina sensorer kan undvika och svänga av från objekt i sin väg. Tack vare sensorns förmåga att upptäcka objekt i avståndet 150-10 cm, kan bilen undvika hinder både på långt och kort håll. På grund av att sensorn så tidigt kan detektera hinder kan den utan att stanna, svänga och avvika från hindret.

Resultatet är en självkörande bil som uppfyller samtliga slutgiltiga mål i kravspecifikationen.

6. SLUTSATS

Projektet gick mestadels som planerat. Majoriteten av målen i kravspecifikationen uppnåddes, men vissa funktioner fick uteslutas för att hinna klart med projektet inom den givna tidsramen. Bilen kan köra av sig själv och undvika hinder vilket var målet med projektet. Dock kan inte bilen upptäcka stup eftersom att vi inte hade tillräckligt med tid att lösa problemet samt att den funktionen var minst viktig för att uppnå målet med en självkörande bil. Bilen kan heller inte backa, men den funktionen behövs inte då bilen kan upptäcka hinder i god tid och därmed svänga undan från alla hinder.

6.1 FRAMTIDA UTVECKLINGSMILJÖER

Projektet skulle kunna utvecklas på flera olika sätt. Ett sätt hade varit genom att implementera en funktion som gör att bilen kan känna av avstånden till hindret så att ifall bilen hamnar för nära ett hinder och inte hade hunnit svänga så kan den backa istället. Då hade även en funktionen att bilen ska kunna backa behöva implementeras.

En annan utvecklingsmöjlighet är att lägga in förutbestämda rutter på bilen så att den kan fungera som transportmedel på olika arbetsplatser. En kombination av förutbestämda rutter samt en sensor som undviker att bilen krockar hade kunnat användas i flera olika arbetsområden som till exempel i ett lagerhus där lådor behöver flyttas till rätt hylla.

7. REFERENSER

I detta avsnitt refereras de källor som har använts till projektet

Microchip Technology, microcontroller - ATmega1284

Sharp Corporation, IR-sensor - SHARP 2Y0A02

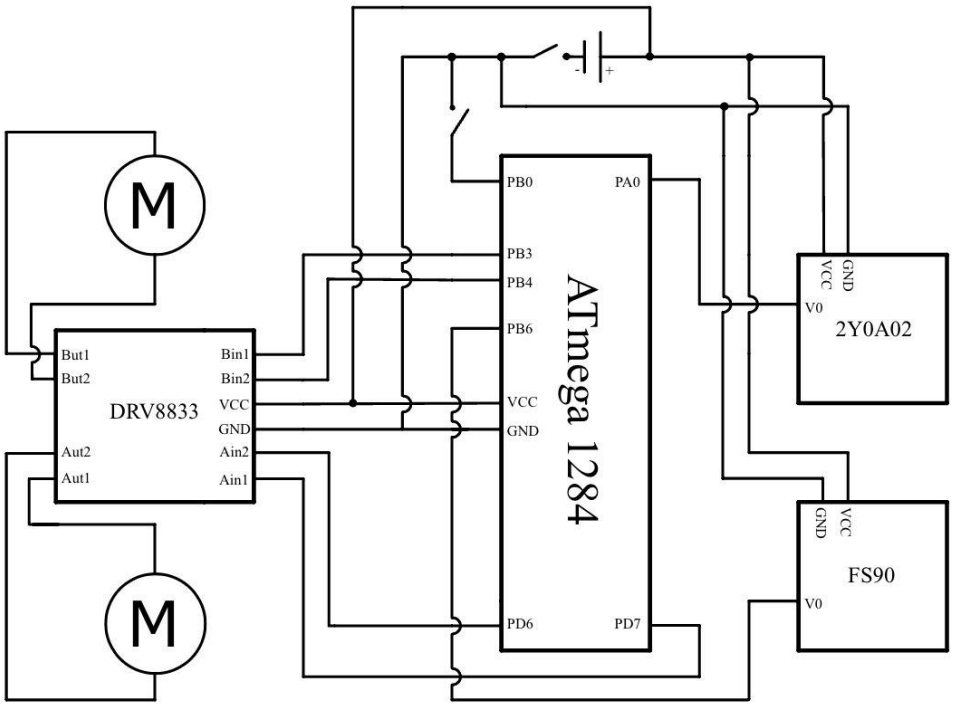
Texas Instruments, Dual H-Bridge motor driver - DUAL H-BRIDGE MOTOR DRIVER - drv8833

FEETECH RC Model CO, Micro servo - FEETECH FS90 micro servo

8. APPENDIX

I detta avsnitt samlas bilagor för projektets kod samt kopplingsschema.

8.1 KOPPLINGSSCHEMA



Figur 1: Kopplingsschema för slutgiltiga kopplingen av självkörande bil.

8.2 C-KOD FÖR PROJEKTET

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#define BUTTON_PIN 0
#define SERVO_CONTROL 6
#define PWM_MOTOR_A1 7
#define PWM_MOTOR_A2 6
#define PWM_MOTOR_B1 3
#define PWM_MOTOR_B2 4

// all i/o in the code collected
void data_direction_init (){
	DDRA = 0b00000000; //adc/sensor
	DDRB &= ~(1 << BUTTON_PIN);		//startbutton
	PORTB |= (1 << BUTTON_PIN);		//referens for startbutton
	DDRD |= (1 << PWM_MOTOR_A1) | (1 << PWM_MOTOR_A2);
	DDRB |= (1 << PWM_MOTOR_B1) | (1 << PWM_MOTOR_B2) | (1 << SERVO_CONTROL);
}
//initializing the AD-conversion
void adc_init(){
	ADCSRA |= (1 << ADEN);
	ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
	ADMUX |= (1 << REFS0);
}
//AD-conversion
```

```
int adc_read(){
	ADCSRA |= (1<<ADSC);
	while(ADCSRA&(1<<ADSC)){}
	return ADC;
}
//reading from the sensor
int read_sensor(){
	if (adc_read() > 250 && adc_read() < 800){
		return 1;
	}
	return 0;
}
//setting the period for the sensor-servo
void set_period_servo(uint16_t period){
	ICR3 = period;
}
//setting the pulse for the sensor-servo
void set_pulse_servo(uint16_t pulse){
	//enligt handledning i labb 2, sätter pulsen genom att sätta output compare register till ett värde
	OCR3A = pulse;
}
//initializes the servo for the sensor
void servo_pwm_init(){
	TCCR3A = 0b10000010;
	TCCR3B = 0b00011010;
	set_period_servo(20000);
	set_pulse_servo(800);
}
//initializes the motor for the right wheel
```

```
void rightWheel_pwm_init(){
// Set Fast PWM mode with TOP = 0xFF, and set OC2A on compare match
        TCCR0A |= (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);
        TCCR0B |= (1 << CS01);
}
//initializes the motor for the left wheel
void leftWheel_pwm_init(){
// Set Fast PWM mode with TOP = 0xFF, and set OC2A on compare match
        TCCR2A |= (1 << COM2A1) | (1 << WGM21) | (1 << WGM20);
        TCCR2B |= (1 << CS21);
}
//set speed for right wheel
void rightWheel_set_speed(int speed){
        OCR0A = speed;
}
//set speed for left wheel
void leftWheel_set_speed(int speed){
        OCR2A = speed; // Set the top value (period) for Timer/Counter2 using OCR2A
}
//driving straight forward
void drive_forward(int speed){
        rightWheel_set_speed(speed);
        leftWheel_set_speed(speed - 5);
}
//when an obstacle, turn depending on where the obstacle is
void change_wheels(int direction, int turn){
        if((direction < 950 && turn < 1) || (direction < 850 && turn > 1)){        //långt höger
                leftWheel_set_speed(70);
        }
```

```
        else if((direction < 1050 && turn < 1) || (direction < 950 && turn > 1)){          //lite höger
                leftWheel_set_speed(0);
        }
        else if((direction < 1200 && turn < 1) || (direction < 1050 && turn > 1)){          //lite vänster
                rightWheel_set_speed(0);
        }
        else if((direction < 1300 && turn < 1) || (direction < 1200 && turn > 1)){          //långt vänster
                rightWheel_set_speed(70);
        }
        _delay_ms(200);
        drive_forward(150);
}

int main(void)
{
        int pulseValue = 800;
        int sensorMovement = 1;   //0 = counterclockwise, 2 = clockwise
        int sensor_buffer = 0;
        data_direction_init();
        adc_init();
        servo_pwm_init();
        rightWheel_pwm_init();
        leftWheel_pwm_init();
        while(!(~PINB & (1 << BUTTON_PIN))){} // waiting for push of button
        drive_forward(130);
        while (1){
                //sensor moving counterclockwise
                for(int i=0;i<4;i++){
                        sensorMovement = 0;
```

```
            pulseValue=pulseValue+110;
            set_pulse_servo(pulseValue);
            for(int i = 0; i<300; i++){
                  read_sensor();
                  if(read_sensor()){            //if obstacle
                        sensor_buffer++;
                  }
                  if(sensor_buffer>10){       //if obstacle still there
                        change_wheels(pulseValue, turn);
                        sensor_buffer=0;
                  }
            }
            sensor_buffer=0;
      }
      //sensor moving clockwise
      for(int i=0;i<4;i++){
            sensorMovement = 2;
            pulseValue=pulseValue-110;
            set_pulse_servo(pulseValue);
            for(int i = 0; i<300; i++){
                  read_sensor();
                  if(read_sensor()){      //if obstacle
                        sensor_buffer++;
                  }
                  if(sensor_buffer>10){       //if obstacle still there
                        change_wheels(pulseValue, turn);
                        sensor_buffer=0;
                  }
            }
```

```
                    sensor_buffer=0;
            }}}
```

Figur 2: Källkod