UPPSALA
UNIVERSITET

TVE 15 034
Examensarbete 15 hp
Juni 2015

Autonom bil med enkapseldator och ultraljudssensorer

tillämpning av en Arduino mikrokontrollerplattform och HC-SR04 sensorer

Carl Bondesson
Erik Stigenius

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Autonomous car with microcontroller and ultrasonic sensors

Carl Bondesson, Erik Stigenius

Autonomous cars or robot cars have been on the agenda ever since Hollywood started with their Sci-Fi genre, maybe even before that. A lot of things have happened since then and now the self-driving vehicle is not far away. In this project, the Autonomous car with microcontroller and ultrasonic sensors, we are looking at a way of making a small, regular RC car autonomous with relatively simple means and investigate how the big companies does it to learn more about the development of the autonomous cars and their technology.

We used an Arduino Uno R3 supplemented with an Arduino Motor Shield R3 as our microcontroller board and three HC-SR04 ultrasonic sensors. By removing almost all of the old parts of the RC-car, except the two DC motors, and replacing them with these new parts we managed to make a vehicle that drove around in a room without crashing into anything. We could have used entirely different sensors, or supplemented the existing setup with other sensors to make it even more accurate and obstacle avoiding. But for our purpose the three ultrasonic sensors did the job. There is always place for optimizing in projects like this one, in our case we could have been optimizing our code more, to make the car perform even better.

Handledare: Svante Andersson
Ämnesgranskare: Ken Welch
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE 15 034

Populärvetenskaplig sammanfattning: Autonom bil

Självkörande bilar är något som fram tills nu endast synts i Hollywood sci-fi block busters som I-Robot och Minority Report. Men de senaste åren har utvecklingen gått snabbt och intresset växt enormt. Flera ledande biltillverkare lovar delvis självkörande bilar redan under 2015. För att hänga med i utvecklingen bestämde vi oss för att bygga vår egen autonoma bil.

Under andra halvan 1900-talet och början av 2000-talet kom funktioner som *cruise control*, *adaptiv cruise control* och *auto parking* i personbilar. I mars i år meddelade den världsledande elbilstillverkaren Tesla Motors, med Elon Musk i spetsen, att deras kunder ska erbjudas delvis självkörande bilar under 2015 [1]. Även organisationen IEEE, idag världens största sammanslutning av professionella elektroingenjörer, ser positivt på utvecklingen och förutspår helt självkörande bilar år 2040 [2].

När vi insåg potentialen och intresset inom detta område bestämde vi oss för att bygga en egen autonom bil. Med tillgång till en välutrustad verkstad hade vi goda praktiska förutsättningar att komma igång. Det vi däremot inte hade var kunskaper inom området. Efter mycket efterforskning om sensorer och minidatorer, bestämde vi oss tillslut för att använda oss av ultraljudssensorer som vi sedan skulle koppla till en minidator från tillverkaren Arduino bestående av Arduino Uno samt Arduino Motor Shield.

Inledningsvis kopplades motorer och sensorer ihop med kontrollkorten, och lämpliga platser att montera sensorerna undersöktes varefter de monterades. För att få motorerna att köra utifrån värden givna av sensorerna, skrevs kod bestående av bland annat algoritmer, som sedan laddades upp på minidatorerna. Resultatet blev en bil som kunde ta sig runt i ett rum med hinder såsom väggar, hörn och stolar, utan att krocka.

Sensorerna har passat bra för ändamålet, trots att de i alltför flacka vinklar gentemot ett hinder inte gett rätt avståndsvärden. Koden har varit tillräcklig avancerad för att vi ska nå vårt mål, men med mer tid hade den gått att förfina väldigt mycket mer, och på så sätt få bilen att prestera bättre. Våra begränsade förkunskaper inom området ledde till att projektet präglades av trial and error. Detta har dock inte varit en nackdel då det i slutändan gett oss grundliga kunskaper om minidatorer, sensorer samt programmering av dessa.

Innehållsförteckning

1 Introduktion

1.1 Historia och utveckling

Enkla typer av autonoma fordon är något som funnits i flera decennier [3]. Kända exempel på dessa är bilar med farthållare samt flygplan och fartyg med autopilot. Systemen har gett flygplan och fartyg möjlighet att automatiskt följa planerade rutter med hjälp av gps, och föraren av en bil har kunnat hålla en konstant hastighet utan att justera med gasen. Vid eventuella hinder eller avbrott i rutten eller på vägen, har de dock fått kopplas bort för att låta föraren manuellt undvika kollisioner. Med andra ord har dessa fordon behövt en förare som aktivt bevakar området omkring fordonet för att upptäcka andra båtar/flygplan/bilar och hinder i allmänhet.

En naturlig utveckling på de klassiska autopiloterna har de senaste åren varit att göra dem smartare, smartare i den meningen att de själva kan upptäcka hinder längs vägen och sedan automatiskt kompensera för dem. Exempel på dessa är bland annat adaptiva farthållare som känner av avståndet till bilar framför och anpassar farten efter det, auto-parkering för bilar, flygplan som kan landa och lyfta helt autonomt och fartyg som automatiskt kan väja för andra fartyg. Med mer avancerade autopiloter har den potential som fullständigt autonoma fordon har, uppenbarats. Detta har drivit på utvecklingen ytterligare och de senaste fem åren har det gått snabbt. Ledande biltillverkare lovar delvis autonoma bilar år 2015, och enligt många experter är helt självkörande bilar realitet på vägarna senast år 2040 [4].

1.2 Tekniken

Tekniken bygger på implementation av radar och ljus-radar (LIDAR), som kan läsa av avstånd, gps, samt olika typer av kameror [5]. För att tolka bilder från kameror används program för bildanalys. Slutligen krävs avancerad mjukvara för att bearbeta data från dessa givare och kombinera den till en bild av omgivningen som autopiloten kan agera utifrån.

1.3 Syfte och målbeskrivning

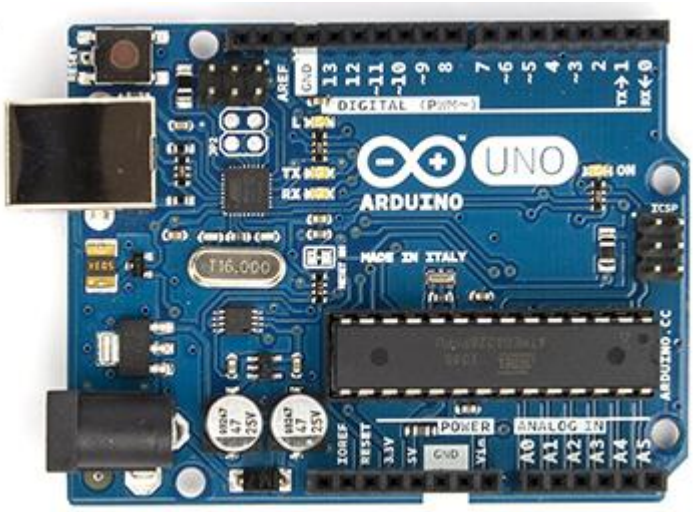
Syftet med projektet är att lära sig hur delar av denna teknik fungerar samt tillämpa den i en radiostyrd bil för att få den autonom. Vidare ska kunskaper inom elektronik, mekanik och programmering användas och vidareutvecklas i den mån som krävs för att nå målet. Målet är att få en vanlig, radiostyrd bil att på egen hand kunna navigera och köra runt i ett rum och undvika kollisioner med hinder såsom väggar, stolar och hörn.

2 Teori

2.1 Arduino mikrokontrollerplattform

En mikrokontroller är en liten enkapseldator som till skillnad från en vanlig dator oftare används för att känna/ta in information från den fysiska världen via till exempel olika sensorer. Sedan kan de via kod och motorer utföra fysiska rörelser grundat på den informationen. Det finns flera olika sorters mikrokontrollers och plattformar till dessa men i det här projektet användes en Arduino Uno R3 då Arduinoplattformen är billig, lätt att förstå för nybörjare samt fungerar bra med olika

operativsystem [6]. En Arduino Uno R3 använder en ATMEL AVR mikrokontroller, mer specifikt ATmega328, och är den tredje generationen av denna sorts mikrokontrollerplattform. Den har 6 analoga pins och 14 digitala pins varav 6 stycken av de digitala är så kallade PWM pins, PWM står för Pulse Width Modulation och är en teknik för att få analoga resultat med digitala medel. PWM pins behövs med andra ord för att kunna kontrollera hur fort motorerna ska spinna, alltså hur mycket el de ska få, och väljs i programmeringskoden med ett tal mellan 0 och 255, där 0 är ingen gas alls och 255 är full gas[7]. För att fungera behöver Arduino Uno 6-20V men det rekommenderade intervallet ligger på 7-12V [8]. Hur en Arduino Uno ser ut översiktligt kan ses i figur 1. Det finns flera olika storlekar på Arduino-plattformarna, till exempel Arduino Mega som har fler pins så att man kan använda fler sensorer, lampor, etcetera eller Arduino Nano som är mer som en liten version av Arduino Uno och passar därför mindre projekt.



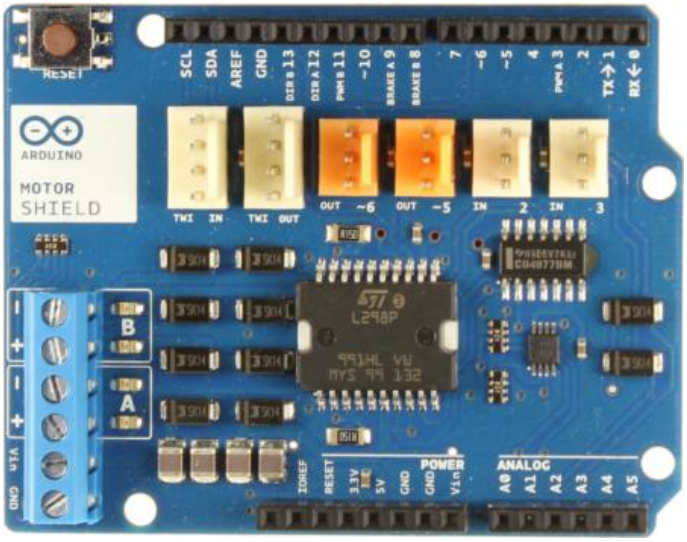
Figur 1: Uppbyggnaden (ovansidan) av en Arduino Uno R3.

2.2 Arduino programmeringsmiljö

Språket som används vid programmering av Arduino-plattformar är skapat för de som inte är så vana vid mjukvaruhantering. Därför är språket ganska enkelt och lätt att lära sig, det skrivs i C/C++ och bygger på de två projekten/språken Processing och Wiring [9]. Koden bygger på två huvudfunktioner, “void setup()” samt “void loop()”. Funktionen setup() körs bara en gång i programmet och det är därför en god idé att initiera variabler i den, detta går dock att göra i egenskapade funktioner eller i början av koden, vilket gör att setup() inte nödvändigtvis behöver användas. Funktionen loop() är till skillnad från setup() ytterst viktig för att det som programmeras ska fungera då den körs om och om igen så länge som plattformen har strömförsörjning. Körningar i programkoden kan förutom att ses rent fysikaliskt (till exempel en lysdiod som blinkar) ses i den Seriella monitorn, i denna monitor kan man skriva ut värden eller texter för att verifiera att koden fungerar.

2.3 Arduino Motor Shield R3

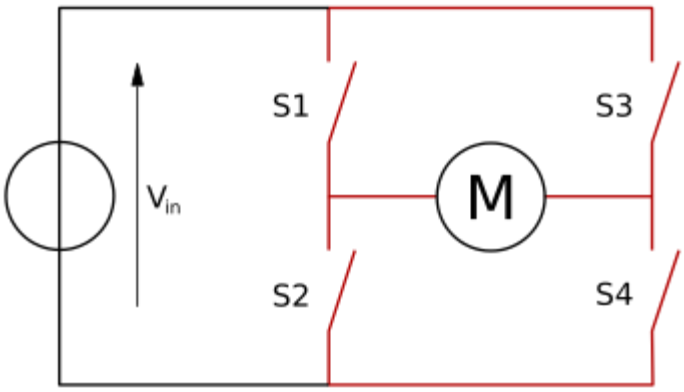
Bilen i projektet hade två stycken likströmsmotorer (en till styrningen och en till vanlig drift) så det behövdes en Arduino Motor Shield R3, se figur 2, som helt enkelt fungerar som en förlängning av mikrokontrollerplattformen och gör att kortet kan styra två motorers riktning och hastighet samtidigt och oberoende av varandra. Detta kort har precis som Arduino Uno ett rekommenderat spänningsintervall in i kortet som ligger på 7-12V [10].



Figur 2: Uppbyggnaden (ovansidan) av en Arduino Motor Shield R3. Kanal A och B, ungefär vid nedre, vänstra hörnet, styr varsin likströmsmotor.

2.4 H-brygga

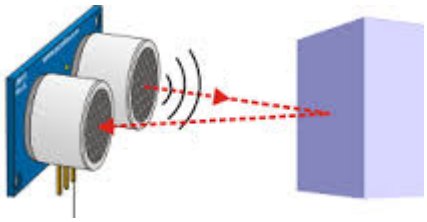
För att bilen ska kunna åka fram och tillbaka samt svänga höger och vänster måste likströmsmotorerna kunna byta riktning, detta sker med hjälp av en H-brygga. En H-brygga är en elektronisk krets som finns monterad på mikrokontrollerplattformen (eller rättare sagt på Motor Shield-kortet) och gör att spänning kan läggas över en last i båda riktningar, se figur 3. Lasten i vårt fall är de två motorerna [11].



Figur 3: En H-brygga består av S1, S2, S3 och S4. S1 och S2 eller S3 och S4 är aldrig stängda samtidigt då det orsakar kortslutning S står för "Switch" (strömbrytare) och M står för "Motor".

2.5 Ultraljudssensorer, HC-SR04

HC-SR04, en sorts ultraljudssensor, används för avståndsmätningar. Ultraljud kallas det ljud som ligger över det frekvensband en människa kan höra, alltså allt ljud över 20kHz. Sensorerna sänder ut ultraljud genom sin transmitter och ljudet färdas rakt fram, med en spridning på 15 grader, tills det stöter på ett objekt varav det sedan studsar tillbaka och tas emot av sensorns mottagare, se figur 4. Om vinkeln mot objektet blir för flack kommer ljudet att reflekteras vidare istället för att reflekteras tillbaka till mottagaren, vilket kan ge felmätningar. Genom att starta en timer när ultraljudet skickas ut kan avståndet beräknas då ultraljudet sprider sig i en hastighet av 340 m/s i luft. HC-SR04 kan mäta avstånd från 2 cm upp till cirka 400 cm med en noggrannhet på 0.3 cm och arbetar med en frekvens på 40kHz. En HC-SR04 behöver en inspänning på 5V för att fungera bra [12].



Figur 4: Den röda pilen visar hur ultraljudet skickas ut från sensorns transmitter, träffar ett objekt och sedan tas emot av mottagaren.

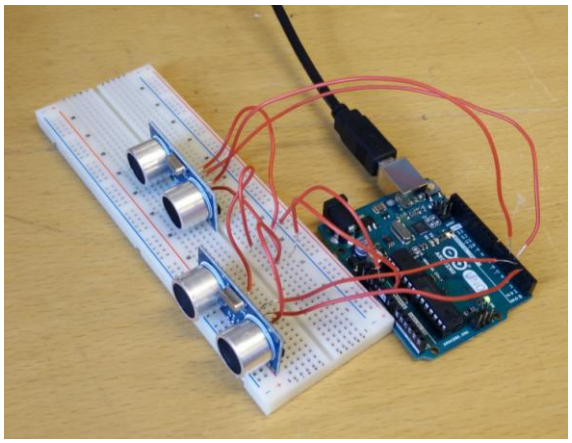
3 Material och metod

Projektet har huvudsakligen bestått av verkstadsarbete i form av lödning och byggande, samt programmering av motorer och sensorer. När problem uppstått har lämplig lösning diskuterats, utförts och slutligen testats. Vid gott resultat har lösningen förfinats ytterligare, i annat fall har en ny lösning tagits fram. Denna "trial and error"-metod har använts såväl i verkstadsarbetet som i

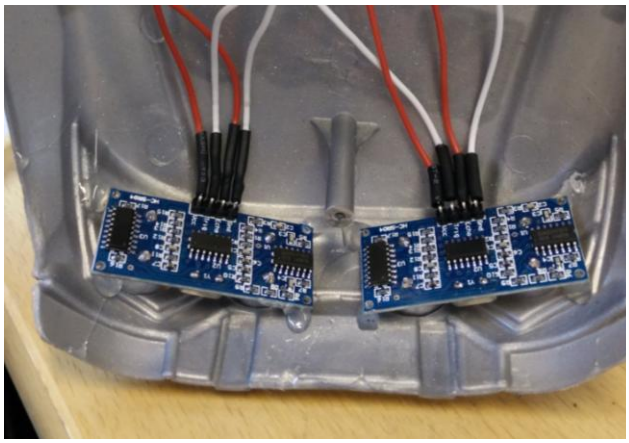
programmeringen på grund av begränsad förkunskap i berörda områden. För slutprodukten har följande komponenter behövts:

- Radiostyrd bil med två likströmsmotorer
- 1 st Arduino Uno R3
- 1 st Arduino Motor Shield R3
- 3 st HC-SR04 ultraljudssensorer
- 16 st 1,5V AA-batterier i batterihållare
- Kablage
- 15cm vinkeljärn i aluminium
- Skruvar och muttrar

Inledningsvis lades stort fokus på verkstadsarbete i form av demontering av bilen. Karossens front förbereddes för montering av sensorer genom att borra hål i passande diameter, bilens original-kontrollerkort plockades ut och bilens likströmsmotorer löddes av. Vidare kopplades de två sensorerna och motorerna in i Arduino Uno med tillhörande Motor Shield för tester (se figur 5), där motor shielden fungerar som en förlängning av huvudplattformen för att addera ytterligare funktioner till kretsen som i detta fall bestod av individuell styrning av likströmsmotorerna. Kretsarna drevs vid denna tidpunkt gemensamt med två stycken 9V-batterier. För att testa motorernas och sensorernas funktioner skrevs testprogrammen *Motortest* samt *Sensortest* (se kod i appendix avsnitt 7.2 och 7.3). Testprogrammet för drivmotorn och styrmotorn testade motorerna med positiv respektive negativ strömriktning samt effekten vid olika strömstyrkor. Sensorerna testades vid olika avstånd, dels en och en, men också parallellt för att se att de gav rätt avstånd.



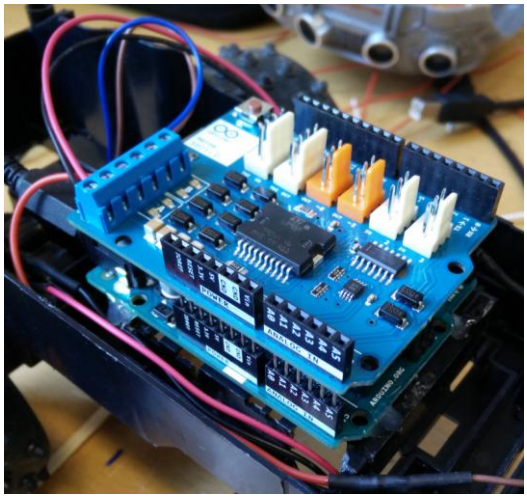
Figur 5: Här syns kretsen vid test av sensorerna.



Figur 6: Visar sensorernas första montering - i karossen.

Vidare löddes sensorerna ihop med kablar (se figur 6) som sedan kopplades ihop med kontrollerkoret. Motorernas kablar kopplades in i varsin kanal på motor shielden (se figur 8). Kretsen monterades sedan på bilen (se figur 7) och programmering av ett huvudprogram inleddes.

Figur 7: Första monteringen av sensorer, sett från utsidan

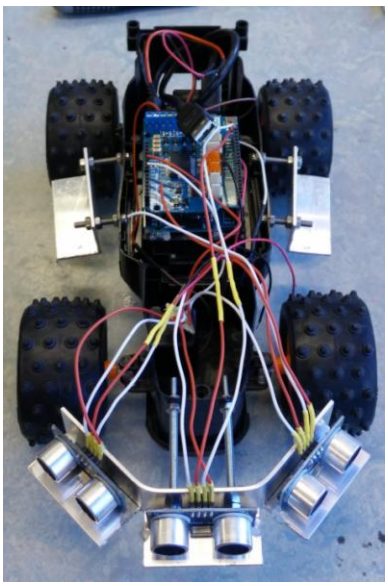


Figur 8: Montering av Motor Shield på Arduino Uno. I hörnet längs upp till vänster syns kanalerna för motorerna.

I kodningen tillämpades, som sagt, “trial and error”-metoden. Koden testades genom att provköra bilen i ett rum med lämpliga hinder såsom väggar, hörn, bord och stolar. Som konsekvens av dåligt resultat vid provkörningen beslutades att ytterligare en sensor skulle monteras.

Lämpliga monteringsplatser undersöktes genom att titta på sensorernas kritiska vinklar gentemot hinder och i figur 9 syns resultatet. Provkörningarna visade även att motorerna fick för lite ström som konsekvens av den gemensamma strömförsörjningen mellan plattformarna. Den gemensamma strömförsörjningen med två 9V-batterier ändrades då till två separata 12V-försörjningar bestående av 8st 1,5V AA-batterier vardera, placerade i taket på karossen (se figur 9) och i utrymmet längst bak, i “bagageluckan” (se figur 10). Den ena till Arduino Uno och sensorer och den andra uteslutande till motorerna. Fördelen med denna uppsättning var att motorerna alltid fick den spänning de behövde för att fungera korrekt. 1,5V-batterierna levererar även högre ström än 9V-batterierna vilket leder till stabilare och jämnare gång för både motorer och sensorer.

Figur 9: Andra monteringen av sensorerna.



Figur 10: Sista monteringen av sensorerna.

Med ny sensor, montering (se figur 9), strömförsörjning och kod blev resultatet bättre, men fortfarande inte tillräckligt bra, och beslut att montera om sensorerna ännu en gång togs. De kritiska vinklarna var vid det här laget redan kända så enda skillnaden i den nya monteringen var framflyttning av de båda sidosensorerna (se figur 10) för att tidigare upptäcka hinder och därav effektivare undvika kollisioner. Slutligen förfinades koden och slutresultatet uppnåddes.

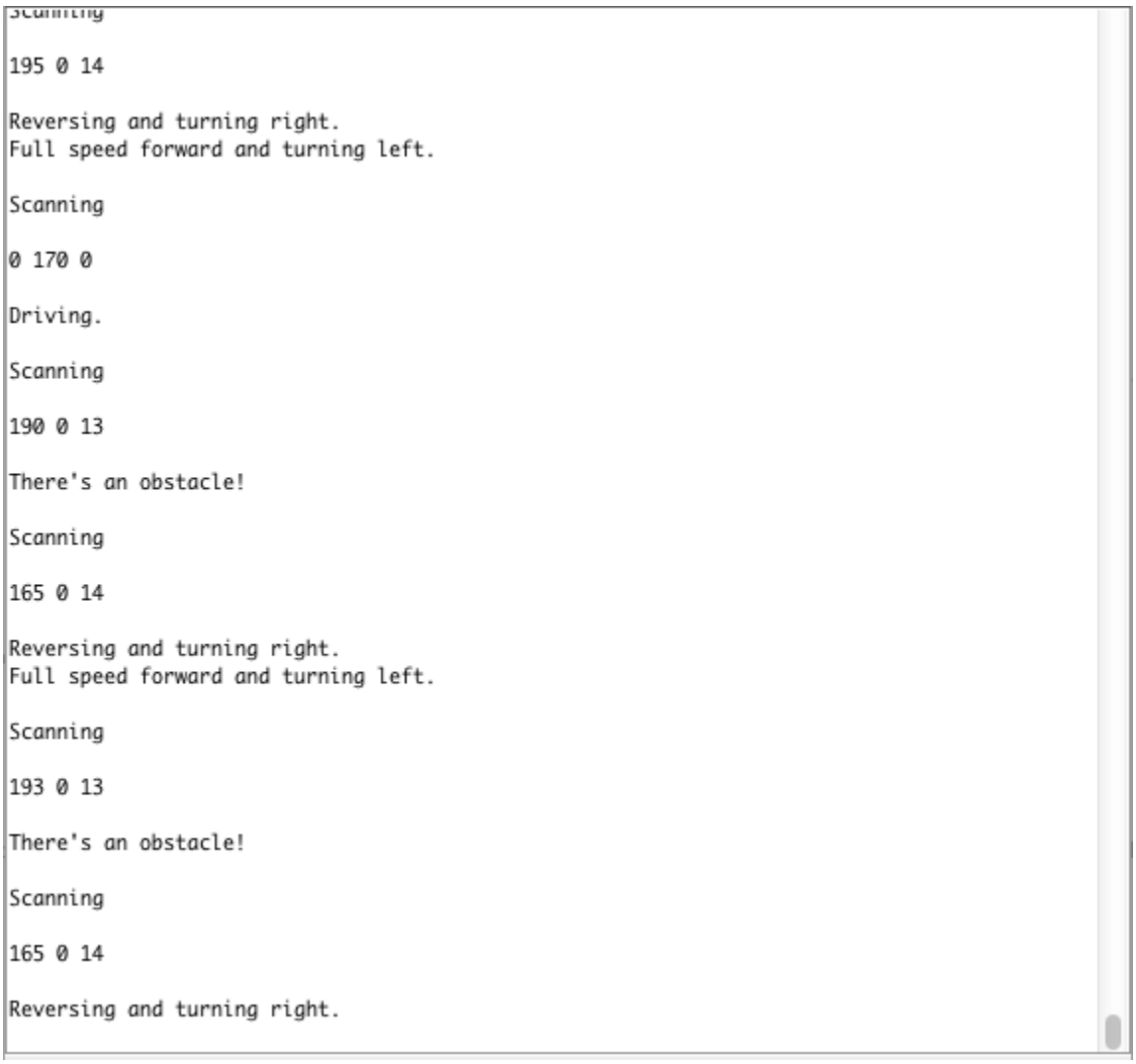
4 Resultat

Slutresultatet blev en bil som kunde åka runt i ett rum utan att kollidera med enklare hinder/objekt såsom väggar, stols- och bordsben samt människor. I figur 11,12 och 13 kan slutresultatet ses rent estetiskt. Huvudkoden som styr den autonoma bilen hittas i appendix avsnitt 7.1. Figur 14 visar ett exempel på hur programmeringsmiljöns monitor kan se ut vid en körning.

Figur 11: Slutgiltig montering av sensorerna

Figur 12: Den färdiga bilen sedd från sidan.

Figur 13: Den färdiga bilen sedd snett uppifrån, man kan ana den ena batterihållare uppe i taket och Arduinokorten under denna.



Figur 14: Resultat av en körning. Avstånden från de tre sensorerna samt vad bilen gör i dessa lägen. Första värdet, 165 längst ner, visar den bilens vänstra sensors avstånd, det mittersta värdet motsvarar den främre sensorn och värdet längst till höger visar den högra sensorns värde.

5 Diskussion

5.1 Sensorplacering och förbättring

I Figur 11 syns den slutliga monteringen av våra tre sensorer. I efterhand när vi jämför de tre olika sensorplaceringarna vi provade kan vi konstatera att den tredje och sista placeringen/monteringen var den bästa. Den täckte in ett större område än den första monteringen och sensorerna upptäckte hinder/objekt tidigare och bättre än vad de gjorde i den andra monteringen. Denna montering lyckades också minska påverkan från situationer vid kritiska/flacka vinklar mest. För att förbättra funktionen ytterligare hade fler sensorer kunnat kopplas in. Detta hade medfört ett ännu större "synfält" för bilen vilket hade varit bra för att den då lättare kunde veta precis vilket håll den skulle svänga eller backa då den upptäckt ett hinder eller objekt. Dock hade fler sensorer medfört mer komplicerad kod och vi hade troligtvis blivit tvungna att använda en annan plattform med fler ingångar/utgångar.

5.2 Begränsande faktorer

Bilens funktion begränsades av en rad olika faktorer. Avsaknandet av regulatorer gjorde att motorerna blev okänsliga. Det fanns inga jämna övergångar mellan olika farter utan endast hög/låg fart och detta ledde till ojämn gång. Även ultraljudssensorerna bidrog med begränsningar. Så kallade spökljud som uppstod av att ultraljudet reflekterades felaktigt i formationer på hindren, ledde till att avståndsavläsningarna blev fel, vilket i sin tur bidrog till bilen agerade fel i vissa tillfällen. Vid flacka vinklar mot hinder studsade ultraljudet inte tillbaka till mottagaren utan istället iväg längs hindret. I dessa situationer kunde bilen stå några cm från ett hinder medan sensorn gav ett stort värde. Detta problem löstes delvis med utökning av antal sensorer samt montering av dem i olika vinklar. I hörn hade ultraljuden en tendens att studsa och reflekteras in i de andra sensorerna, vilket också gav fel värden på avstånd och ställde till med problem för bilen i försök att ta sig därifrån. Slutligen var koden en begränsning. Självklart hade det gått att lägga många timmar till på att ta fram de bästa algoritmerna för självstyrning, men på grund av tidsbegränsningen var detta inte möjligt.

5.3 Ultraljudssensorer kontra IR-sensorer

Förutom fler sensorer hade vi även kunnat använda andra sorters sensorer. Ett bra substitut för ultraljudssensorer är IR-sensorer, som vi hade tänkt använda från början. Vi valde att använda ultraljudssensorer då vi bestämde oss för att använda ett lektionsrum eller en korridor som försöksplats. Eftersom en IR-sensor använder ljus när den mäter avstånd går visserligen mätningarna fort, men mätningarna kan ge olika resultat i ett rum med objekt i olika färger då ljus reflekteras olika beroende på färg, nyans och skuggor, även fast objekten ligger på precis samma avstånd. Vi ville även att bilen skulle kunna användas ute och där fungerar inte en IR-sensor så bra heller på grund av bakgrundsstrålningen. Visst, ultraljudssensorerna har också sina nackdelar som nämnt innan, men när vi ställde sensorernas funktioner mot varandra och

jämförde dessa med vad vi ville uppnå för resultat, kom vi fram till att ultraljudssensorer skulle fungera bäst.

5.4 Ström- och spänningsförsörjning av komponenterna

Något vi märkte spelade stor roll på motorernas och sensorernas funktioner var spännings- och strömförsörjningen. Vid gemensam försörjning av alla komponenter från 2x9V-batterier, delades spänningen upp på ett oönskat sätt. Sensorerna, som behövde 5V var, fick betydligt mer än så i onödan, och motorerna som behövde minst 5,5V var, fick betydligt mindre. Den låga spänningen över motorerna gjorde att de inte fick tillräckligt med kraft för att fungera korrekt. Med ett 8x1,5V-batteripack kopplat direkt till motorerna fick de däremot tillräckligt med ström och spänning. Sensorerna och Arduino Uno matades med ett till likadant batteripack, vilket gav sensorerna ca 5V vardera. Denna lösning gjorde att alla komponenter fick tillräckligt med spänning och ström för att fungera bra. Det enda negativa med att byta ut de två 9V-batterierna mot de två stora batteripaketen var att bilen blev ganska mycket tyngre, effekten från de nya batteripaketen vägde dock upp för det.

6 Slutsatser

Vi lyckades bygga en liten bil som kunde åka runt av sig själv utan att krocka med större föremål. Det finns alltid utrymme för utveckling och optimering i projekt där något som “tänker själv” ska skapas. I vårt fall har det inneburit oändlig förbättring av algoritmer i koden. Därför är det viktigt att ha ett tydligt mål med hur man vill att slutprodukten ska vara. De metoder vi använt har fungerat bra med hänsyn till vårt mål. Sensorerna har fungerat bra för ändamålet även om andra typer av sensorer som komplettering hade kunnat ge ett bättre resultat.

7 Referenser

[1] Karl Johan Byttner, Veckans Affärer, 20/03/15. http://www.va.se/nyheter/2015/03/20/sjalvkorande-bilar-redan-i-sommar/. 27/05/15

[2] Francine Tardo, Monika Stickel, IEEE, 02/09/12. http://www.ieee.org/about/news/2012/5september_2_2012.html. 26/05/15

[3] "Carnegie Mellon". Navlab: The Carnegie Mellon University Navigation Laboratory. The Robotics Institute. 26/05/15.

[4] Francine Tardo, Monika Stickel, IEEE, 02/09/15. http://www.ieee.org/about/news/2012/5september_2_2012.html. 26/05/15

[5] G. Meyer and S. Deix, "Research and Innovation for Automated Driving in Germany and Europe," i Road Vehicle Automation, Berlin, Springer, 2014.

[6] http://www.arduino.cc/en/Guide/Introduction. Arduino.

[7] http://www.arduino.cc/en/Tutorial/PWM. Arduino.

[8] http://www.arduino.cc/en/Main/ArduinoBoardUno. Arduino.

[9] http://www.arduino.cc/en/Guide/Introduction. Arduino.

[10] http://www.arduino.cc/en/Main/ArduinoMotorShieldR3. Arduino.

[11] Chuck McManis. http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/. 23/12/06.

[12] http://www.kjell.com/sortiment/el/elektronik/mikrokontroller/arduino/avstandssensor-p87891. Kjell & Company.

8 Appendix

8.1 Huvudkod

```
#include <NewPing.h>            //Includes the library NewPing which is needed for the sensors to work good.

#define SONAR_NUM     3         //Number of sensors.
#define MAX_DISTANCE 300        //Maximum distance to ping (cm).

   //Defining variables, because of the "unsigned" they can only be positive.
unsigned int frontDist;
unsigned int leftDist;
unsigned int rightDist;
unsigned int Time1;
unsigned int Time2;
unsigned int Time3;

NewPing sonar[SONAR_NUM] = {        //Sensors object array.
  NewPing(3, 2, MAX_DISTANCE),      //Each sensor's trigger pin, echo pin, and max distance to ping.
  NewPing(5, 4, MAX_DISTANCE),
  NewPing(7, 6, MAX_DISTANCE),
};

void setup() {

  Serial.begin(115200);             //Sets the data rate to max.

  //Setup Channel A TURN
  pinMode(12, OUTPUT);              //Initiates Turn Channel A pin.
  pinMode(9, OUTPUT);               //Initiates Brake (steer straight) Channel A pin.

  //Setup Channel B THROTTLE
  pinMode(13, OUTPUT);              //Initiates Throttle Channel B pin.
  pinMode(8, OUTPUT);               //Initiates Brake Channel B pin.

}

void turnLeft() {
  //Motor A LEFT
  digitalWrite(12, HIGH);           //Establishes left direction of Channel A.
  digitalWrite(9, LOW);             //Disengage the Brake for Channel A.
  analogWrite(3, 255);              //Spins the motor on Channel A at full speed – maximum left turn.
}

void turnRight() {
```

```
  //Motor A RIGHT
  digitalWrite(12, LOW);          //Establishes right direction of Channel A.
  digitalWrite(9, LOW);           //Disengage the Brake for Channel A.
  analogWrite(3, 255);            //Spins the motor on Channel A at full speed – maximum right turn.
}

void drive() {
  //Motor B FORWARD
  digitalWrite(13, HIGH);         //Establishes backward direction of Channel B.
  digitalWrite(8, LOW);           //Disengage the Brake for Channel B.
  analogWrite(11, 100);           //Spins the motor on Channel B less than half full speed.
}

void fullSpeed() {
  //Motor B FORWARD
  digitalWrite(13, HIGH);         //Establishes backward direction of Channel B
  digitalWrite(8, LOW);           //Disengage the Brake for Channel B
  analogWrite(11, 200);           //Spins the motor on Channel B at ~80% speed

}

void reverse() {
  //Motor B BACKWARDS
  digitalWrite(13, LOW);          //Establishes forward direction of Channel B
  digitalWrite(8, LOW);           //Disengage the Brake for Channel B
  analogWrite(11, 200);           //Spins the motor on Channel B at ~80% speed
}

void brake() {
  //Motor B BACKWARDS
  digitalWrite(13, LOW);          //Establishes forward direction of Channel B
  digitalWrite(8, LOW);           //Disengage the Brake for Channel B
  analogWrite(11, 255);           //Spins the motor on Channel B at full speed
  delay(400);
  slowDown();
}

void slowDown() {
  digitalWrite(8, HIGH);          //Engage the Brake for Channel B
}

void steerStraight() {
  digitalWrite(9, HIGH);          //Engage the Brake for Channel A
}

void loop() {
  scan();                         //Goes to the scan function.
```

```
 if (frontDist == 0 || frontDist >= 40)    //The sensors ping 0 if the distance is greater than the  MAX_DISTANCE.
 {
  if (rightDist <= 40 || leftDist <= 40)
  {
   if (rightDist == 0 || leftDist == 0) { //If the distance is far at the left or the right direction, go forward.
    steerStraight();
    drive();
    Serial.println("Cruising.");
   }
   else {                          //If something appears at the side of the car, brake and then use the navigate function.
    slowDown();
    navigate();
   }
  }

  else {                          //If the distance is further than 40cm at every direction, go forward.
   steerStraight();
   drive();
   Serial.println("Cruising.");
  }
 }

 else                                  //Else (if there is something infront of the robot within 40cm) then slow down and
navigate.
 {
  brake();
  navigate();
 }
}

void scan()                                //This function scans the surroundings with the ultra sonic sensors and determines
the distances in three directions.
{                                     //It also prints the three distances, to the left, to the right and to the front, in the Seriell
Monitor.
 Time1 = sonar[0].ping();
 Time2 = sonar[1].ping();
 Time3 = sonar[2].ping();
 leftDist = Time1 / US_ROUNDTRIP_CM;
 frontDist = Time2 / US_ROUNDTRIP_CM;
 rightDist = Time3 / US_ROUNDTRIP_CM;
 delay(50);
 Serial.println("");
 Serial.println("Scanning");
 Serial.println("");
 Serial.print(leftDist);
 Serial.print(" ");
```

```
  Serial.print(frontDist);
  Serial.print(" ");
  Serial.print(rightDist);
  Serial.println(" ");
  Serial.println(" ");
}

void navigate()                                  //This function determines which way the car should go depending on the three
distances from the function scan and then prints
{                                                //which way the car goes in the Seriell Monitor.
  Serial.println("There's an obstacle!");
  scan();

  if (frontDist < 20) {

    if (rightDist < leftDist) {
      reverse();
      turnRight();
      Serial.println("Reversing and turning right.");
      delay(1000);
      fullSpeed();
      turnLeft();
      Serial.println("Full speed and turning left!");
      delay(1000);
    }

    else if (rightDist > leftDist) {
      reverse();
      turnLeft();
      Serial.println("Reversing and turning left.");
      delay(1000);
      fullSpeed();
      turnRight();
      Serial.println("Full speed and turning right!");
      delay(1000);
    }
  }
  else if (frontDist >= 20) {

    if (rightDist < leftDist) {
      fullSpeed();
      turnLeft();
      Serial.println("Full speed and turning left!");
      delay(500);
    }

    else if (rightDist > leftDist) {
```

```
      fullSpeed();
      turnRight();
      Serial.println("Full speed and turning right!");
      delay(500);
    }
  }
}
```

8.2 Sensortest

– Orginalkoden[1] var gjord för 15 sensorer, vi modifierade denna till våra 3 sensorer.

```
#include <NewPing.h>

#define SONAR_NUM     3 // Number or sensors.
#define MAX_DISTANCE 40 // Maximum distance (in cm) to ping.
#define PING_INTERVAL 33 // Milliseconds between sensor pings (29ms is about the min to avoid cross-sensor echo).

unsigned long pingTimer[SONAR_NUM]; // Holds the times when the next ping should happen for each sensor.
unsigned int cm[SONAR_NUM];         // Where the ping distances are stored.
uint8_t currentSensor = 0;          // Keeps track of which sensor is active.

NewPing sonar[SONAR_NUM] = {     // Sensor object array.
  NewPing(3, 2, MAX_DISTANCE), // Each sensor's trigger pin, echo pin, and max distance to ping.
  NewPing(5, 4, MAX_DISTANCE),
  NewPing(7, 6, MAX_DISTANCE),

  //    NewPing(9, 8, MAX_DISTANCE), // Each sensor's trigger pin, echo pin, and max distance to ping.
  //  NewPing(11, 10, MAX_DISTANCE),
};

void setup() {
  Serial.begin(115200);
  pingTimer[0] = millis() + 75;           // First ping starts at 75ms, gives time for the Arduino to chill before starting.
  for (uint8_t i = 1; i < SONAR_NUM; i++) // Set the starting time for each sensor.
    pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;

}

void loop() {
  for (uint8_t i = 0; i < SONAR_NUM; i++) { // Loop through all the sensors.
    if (millis() >= pingTimer[i]) {         // Is it this sensor's time to ping?
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will be pinged.
      if (i == 0 && currentSensor == SONAR_NUM - 1) oneSensorCycle(); // Sensor ping cycle complete, do something
with the results.
```

[1] http://playground.arduino.cc/Code/NewPing. arduino.cc.

```
      sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before starting a new ping
(insurance).
      currentSensor = i;                          // Sensor being accessed.
      cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo for this sensor.
      sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues, interrupt will call echoCheck to
look for echo).
    }
  }
}

void echoCheck() {                                // If ping received, set the sensor distance to array.
  if (sonar[currentSensor].check_timer())
    cm[currentSensor] = sonar[currentSensor].ping_result / US_ROUNDTRIP_CM;
}

void oneSensorCycle() {                           // Sensor ping cycle complete, do something with the results.
  for (uint8_t i = 0; i < SONAR_NUM; i++) {
    Serial.print(i);
    Serial.print("=");
    Serial.print(cm[i]);
    Serial.print("cm ");
  }
  Serial.println();
}
```

8.3 Motortest

```
void setup() {
  //Setup Channel A - turning
  pinMode(12, OUTPUT); //Initiates Motor Channel A pin
  pinMode(9, OUTPUT); //Initiates Brake Channel A pin

  //Setup Channel B - forward/backward
  pinMode(13, OUTPUT); //Initiates Motor Channel A pin
  pinMode(8, OUTPUT);  //Initiates Brake Channel A pin
}

void loop(){

  //Motor A forward @ full speed
  digitalWrite(12, HIGH); //Establishes forward direction of Channel A
  digitalWrite(9, LOW);   //Disengage the Brake for Channel A
  analogWrite(3, 255);   //Spins the motor on Channel A at full speed
  delay(2000);

  //Motor B backward @ half speed
  digitalWrite(13, LOW);  //Establishes backward direction of Channel B
  digitalWrite(8, LOW);   //Disengage the Brake for Channel B
```

```
  analogWrite(11, 123);    //Spins the motor on Channel B at half speed
  delay(3000);
  digitalWrite(9, HIGH);  //Engage the Brake for Channel A
  digitalWrite(9, HIGH);  //Engage the Brake for Channel B
  delay(1000);

  //Motor A forward @ full speed
  digitalWrite(12, LOW);  //Establishes backward direction of Channel A
  digitalWrite(9, LOW);   //Disengage the Brake for Channel A
  analogWrite(3, 123);    //Spins the motor on Channel A at half speed
  delay(2000);

  //Motor B forward @ full speed
  digitalWrite(13, HIGH); //Establishes forward direction of Channel B
  digitalWrite(8, LOW);   //Disengage the Brake for Channel B
  analogWrite(11, 255);   //Spins the motor on Channel B at full speed
  delay(3000);
  digitalWrite(9, HIGH);  //Engage the Brake for Channel A
  digitalWrite(9, HIGH);  //Engage the Brake for Channel B
  delay(1000);
}
```