YRGO

2022

JESKO OCH AGERA RS

Rapportförfattare:	Daniel Mentzer
Utförare:	Daniel Mentzer, Jacob Nilsson
Handledare:	Erik Pihl, Hampus Ram
Datum laboration:	2022-05-17
Datum rapport:	2022-06-05

SAMMANFATTNING

Detta projekt är den del av Yrgos elektronikingenjörsutbildning, syftet är att jobba projektorienterat och resultatinriktat, och få förståelse för hur de olika faserna i ett projekt påverkar slutresultatet, som i det här fallet var att konstruera en autonom bil och tävla mot andra projektgrupper från Yrgo.

Bilen skulle designas och uppfylla kraven för att tävla i folkrace-kategorin i robot-SM, till sitt förfogande fick varje grupp en förbestämd summa för inköp och 750h per grupp.

Slutresultatet blev två tävlingsklara bilar som placerade sig på första och andra plats på tävlingsdagen.

INNEHÅLL

1. INTRODUKTION

Målet var att konstruera en autonom bil som skulle kunna tävla i ett folkrace med tre andra bilar på banan samtidigt. Vi valde att planera och utföra projektet enligt den agila metoden, vilket innebär att man jobbar i kortare etapper med ett antal delmål. Det här projektet gjorde vi också för att få erfarenhet och kunskap om fördelarna, men också svårigheterna, som kommer av att vara flera personer som jobbar i grupp för att nå ett gemensamt slutmål.

Bilen skulle konstrueras efter reglerna som gäller för folkrace-klassen under robot-SM, och det är framför allt storleksbegränsningen man behöver ta hänsyn till, där bilen inte får vara över 150mm bred, 200mm lång och 150mm hög eller väga över 1kg. Bilen får heller inte konstrueras för att skada eller förstöra en annan bil eller funktionär som kan behöva plocka upp bilen från banan.

Utöver reglerna för tävlingen så fick alla grupper en budget på 8000kr som skulle täcka alla kostnader för projektet, samt en tidsram på 750h per grupp fördelat på fem gruppmedlemmar, och en deadline som var på tävlingsdagen.

2. PROJEKTSTART

Vid projektets start valde vi roller för respektive gruppmedlem, Jacob Nilsson blev projektledare, Daniel Mentzer ansvarig för att kod och komponenter skulle fungera ihop, Ahmed Kopic och Parviz Wanitwijan blev ansvariga för design och Jaffar Naem blev ansvarig för inköp och ekonomi.
Ett avtal upprättades med ett datum för deadline, vår budget i form av arbetstimmar samt materialkostnader.

Därefter började planeringen för utförandet av projektet, de största frågorna var:
Vilka komponenter ska användas?
Vad ska syftet med varje komponent vara?
Vilka funktioner ska bilen ha?
Hur ska tidsplanen se ut?

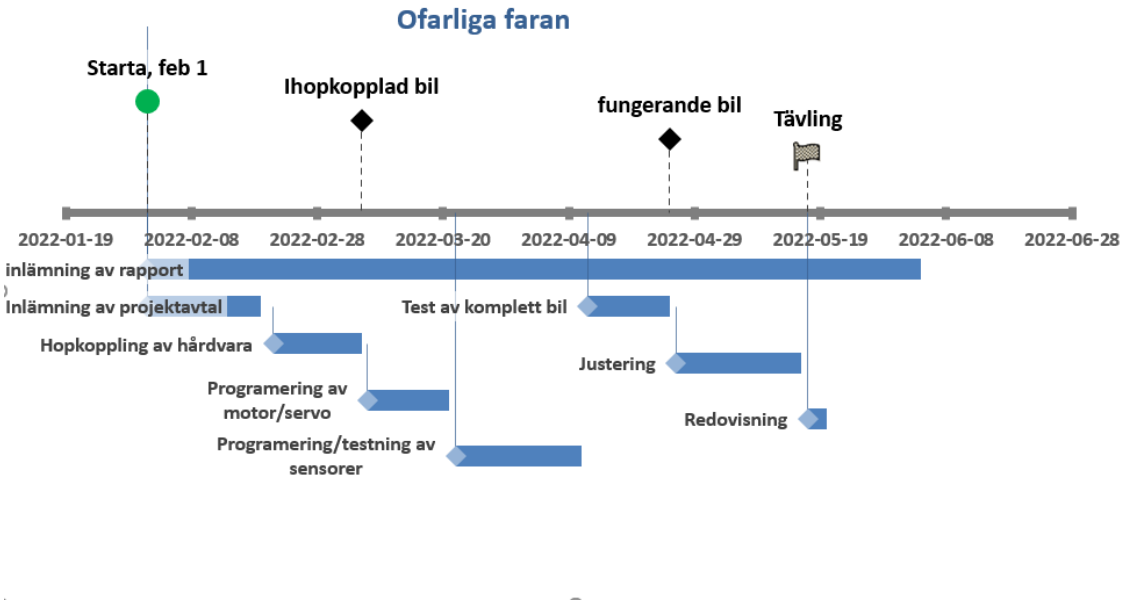
2.1 PLANERING

Vi som grupp ville jobba enligt den Agila metoden, vilket innebär att man sätter upp delmål och arbetar parallellt med olika mindre projekt samtidigt.

Vi delade upp projektet i fyra delar med tre delmål och deadlines. De första två delarna innefattade planering och hårdvara, där delmålet var att ha konstruerat en bil med komplett hårdvara. Andra delmålet var att bilen skulle ha en färdig kod att utgå ifrån. Tredje och sista delmålet var en färdig bil, med deadline på tävlingsdagen, vi använde den sista perioden till mindre justeringar och att eventuellt lägga till fler funktioner.

Redan under planeringen stötte vi på vårt första stora problem, brist på engagemang. Det framkom tydligt att tre av de fem personerna inte ville vara delaktiga, två av dem hoppade av utbildningen i sin helhet och den tredje ansåg sig inte ha tid att delta i projektet. Alltså fick vi snabbt frångå vår ursprungliga plan att arbeta parallellt med projektet uppdelat, och i stället bara försöka hålla oss till våra tre stora delmål.

Det här satte givetvis press på oss som var kvar att arbeta snabbare och lägga mer tid på projektet per person än vad vi ursprungligen räknat med. För att reda ut det valde vi att göra alla moment ihop, vi avslutade varje lektion med att sätta nya mål till nästa, det gjorde att vi fick lättare att dela upp uppgifterna mellan oss samtidigt som vi kunde ta hjälp av varandra när vi stötte på utmaningar och problem.



2.2 TIDSÅTGÅNG

	Planering	Hårdvara	CAD	Programmering	Max
Daniel	25h	40h	40h	45+h	150+h
Jacob	30h	40h	35h	35h	150h
Andra	1h	0h	0h	0h	

2.3 RISKANALYS

HÖG RISK STOR PÅVERKAN	HÖG RISK LÅG PÅVERKAN
1. För mycket tester kan göra att delar går sönder 2. Försenade leveranser	1. Mindre krock 2. Felprogrammering 3. Felsignal av sensor
LÅG RISK STOR PÅVERKAN	**LÅG RISK LÅG PÅVERKAN**
1. Kortslutning 2. Komponenter släpper från konstruktionen 3. Komponentfel (kan vara svårt att upptäcka) 4. Klarar ej av hinder på banan 5. Batterier dör	1. Spöksignaler

2.4 BUDGET/BESTÄLLNINGAR

Beskrivning	Antal	Pris per styck	Fått antal
Distansmätare	3	107,51	3
Ultrasonic sensor	5	133,79	
Arduino Nano 2040	3	223,26	3
Arduino Nano	2	173,45	
H-brygga	4	47,20	4
Batteri	2	175,20	1
Avståndsmätare	1	263,20	1
Styrled	4	23,20	1
Styrservo	2	180	1
Adapter	1	31,20	1
Batteriladdare	1	319,20	1
Kontakter	2	327	2
Kontakter	1	116	1
Sensor	3	151	3
Kontakter	10	6	10
Batteri	1	111	4
Lödtenn	1	76	1
Mikrodator	1	872	1
Kylfläkt	1	46,85	1
Skruvmejslar	1	693,40	1
Mikrodator	4	127	4
SD-kortläsare	1	54,39	2
SD-kort	2	92,80	2
		Beställt:	**Levererat:**
		7444 kr	6068,21 kr

2.5 KOMPONENTER

Det som gjordes först, och som kom att göra oss väldigt framgångsrika, var våra val av komponenter, där vi först och främst utgick från att vi ville ha en bil som skulle fungera vid tävlingsdagen.

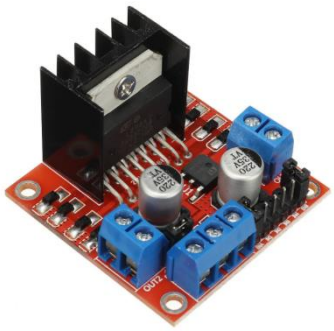
Vi ville ha en bil med bra förutsättningar för att montera på eftermarknadsdelar som till exempel nytt servo, eventuellt en ny motor, och även bra med plats för att kunna montera mikrodator och sensorer. Vi fick tag på en Summon swift som uppfyllde alla storlekskrav, och inte heller hade massa onödig plast som behövde modifieras för att få plats med nya delar.

Komponenter som behövs för att göra bilen autonom är dels en mikrodator. Här gjorde vi valet att börja med en som alla kände sig bekväm med då vi ansåg att den största utmaningen med detta projekt var att få en bil med en kod som var bättre än de andra bilarnas, därför valde vi Arduino nano, som bygger på mikroprocessorn Atmega328p [1] som alla redan har arbetat med vid tidigare projekt. Vi hade ambitioner att använda en annan Arduino mikrodator som bygger på Raspberrys nya mikroprocessor rp2040, men detta kom vi överens om att det sker när det redan finns en färdig bil som vi i så fall väljer att vidareutveckla.

Nästa stora val var vilken typ av sensor som skulle användas, här stod det mellan en modell som kommunicerar med det digitala protokollet i2c, vilket ingen i gruppen tidigare använt, eller en analog modell som ger en spänning beroende på avstånd mellan sensor och föremål framför sensorn.

Sensorn som valdes var Sharp GP2Y0A21YK0F [2] och kom i flera olika varianter med hur långt bort de kunde avläsa ett föremål, här testade vi två olika varianter, den som läste mellan 4 cm och 30 cm och den som läste mellan 10 cm och 80cm, det var det senare alternativet som till slut blev det vi använde då den visade sig fungera bättre för vårt ändamål.

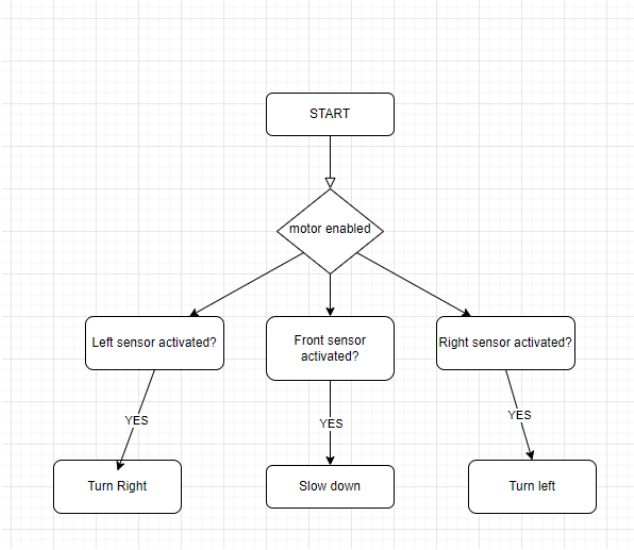
Då bilen redan kom med en motor och ett servo för styrningen så behövdes bara en H-brygga för att kunna leverera tillräckligt med ström för vår borstade motor, detta då utgångarna på Arduinon som mest kan leverera 40ma, medan motorn mättes upp till att dra nästan 1A vid 8V. Modulen som valdes var L298N [3] som hade två utgångar som kunde leverera upp till 1A vardera, vilket passade vår motor bra, den var dessutom billig så flera kunde beställas ifall olyckan



skulle vara framme.

Vi gjorde valet att byta ut servot mot ett som var starkare och snabbare, ett Savöx sh-0257mg [4], dels för att det hade ett datablad som gjorde att vi förstod hur koden för servot skulle se ut men främst för att det var rätt storlek och från ett erkänt bra märke, alltså skulle det med största sannolikhet inte gå sönder.

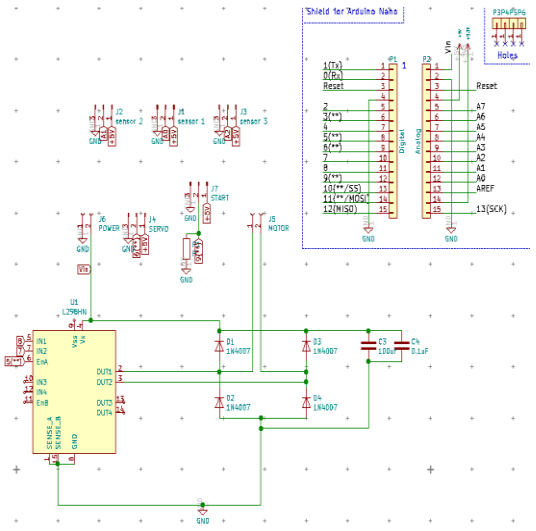
2.6 BILENS FUNKTIONER



Första versionen hur programmet för bilen skulle fungera.

En klurig men viktig del i detta projekt var att i ett tidigt stadie ta fram en bra modell för hur bilen skulle fungera, framför allt vad gäller sensorerna, dels hur många, dels deras placering. En idé var att ha en sensor framåt, två sensorer i 45 graders vinkel och två i 90 graders vinkel, detta främst för att kunna avgöra om bilen var nära en vägg eller en annan bil, då en vägg påverkar fler sensorer än vad en annan bil gör.
Men beslutet blev att bara använda tre sensorer, en framåt som skulle känna av en vägg och då sakta ner för att svänga, och en enkel variant där den svänger höger om vänster sensor läser att den är närmre en vägg än den högra. Den här modellen är väldigt nära det slutgiltiga programmet vilket ingen i gruppen förväntat sig.

3 BYGGA EN BIL



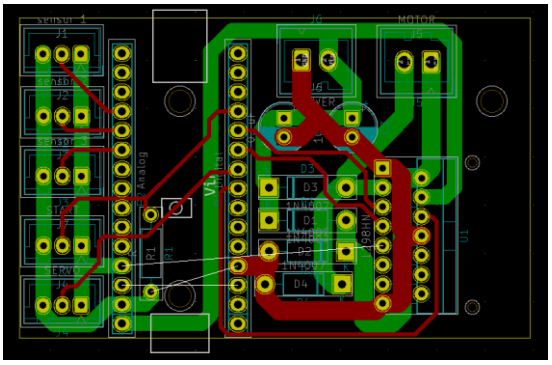
Första kortet som designades i KiCad

Vi delade upp projektet i tre delmål där det första målet var att få ihop en bil där all hårdvara är monterad och ihopkopplad. Vi började med att testa varje komponent för sig med en väldigt enkel bit kod, till sensorn valde vi en kod som vi tidigare använt för en temperatursensor, och till motorn och servot använde vi en kod vi vid ett tidigare projekt programmerat en pwm-funktion med.

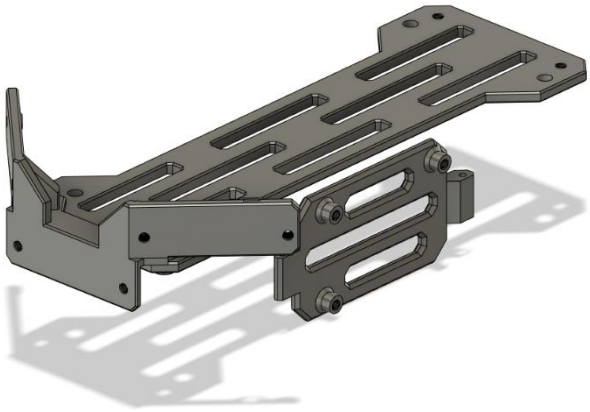
När alla komponenter var testade och en tydlig bild av slutresultatet fanns designade vi ett nytt kretskort att montera mikrodatorn på. Kortet vi beställt innehöll komponenter som inte behövdes för bilen och var samtidigt ganska klumpigt, så i stället för att behöva använda oss av breadboards med risk för kablar som lossnar eller delar som är i vägen, designade vi ett eget kort med en H-

brygga kopplad till ett Arduino, plats för inkoppling av tre sensorer, ett servo och en startknapp.

En risk i att göra så som vi gjorde var att om något blivit fel eller behövde ändras skulle detta sannolikt bli svårare att lösa, men i och med att alla våra kopplingar och komponenter testats i förväg kändes risken väldigt låg.



Det andra steget i att få en färdig bil var att ha någonstans att montera alla delar, det visade sig bli väldigt enkelt att ordna då en av gruppmedlemmarna har en 3D-skrivare, något som kom att bli ovärderligt för hela projektet. Vi designade en platta för att kunna montera delar tillfälligt på och prova oss fram, på denna monterade vi även en hållare för de tre sensorerna, och en hållare för kretskortet vi gjort själva, det här gjorde att bilen snabbt tog form och tidsplanen för att ha en färdig bil inte blev några problem att hålla.



Första plattformen som designades för montering av tre sensorer, ett kretskort och en platta att bygga vidare på.

3.1 PROGRAMMERA BIL

Här började den riktiga utmaningen, då ingen i gruppen hade någon speciell erfarenhet av programmering sen tidigare var detta den svåraste biten. Koderna vi använt för att testa motorn och servot hade en stor brist, de använde en fördröjningsfunktion som i princip pausar hela programmet under en bestämd tid, och detta fungerar bra så länge det bara är en komponent ska användas med en kod som enbart styr denna, men så fort någonting annat ska göras av programmet kommer det fungera bristfälligt eftersom programmet då är pausat. För att lösa problemet behövde vi styra våra delar med varsin timer, något som kom att bli det som krånglade mest i vårt program av olika anledningar.

Vi bestämde oss även för att testa på ett nytt språk, tidigare erfarenheter hade varit framför allt C, men C++ klasser lockade väldigt mycket så vi bestämde oss för att testa på det.

Att skriva koden för bilen var det mest tidskrävande i detta projekt, men också det mest lärorika och roliga.

Det var framför allt koden som krånglade mest, i vår första version visade det sig att vi hade ett fel, och efter mycket felsökande, letande och väldigt lite förkunskap i just felsökning, så hittade vi till slut att en ”enable”-variabel för motorns timer i koden

sattes till "false" utan att vi någon gång tilldelade den det.
På grund av att vi inte hittade vart denna variabel ändrades valde vi att slänga den koden och börja om. Detta gjordes för att ej få samma problem igen. Den nya versionen fungerade bra, och bara några dagar efter vårt uppsatta mål för en färdig bil med en färdig kod kunde den provköras.

Provkörningen blev, enligt oss, ett fiasko! Bilen tog sig förvisso ett varv runt banan, men det var på håret, och bilen var absolut inte förutsägbar.
Problemet var att bilens servo hoppade väldigt mycket fram och tillbaka, det såg nästan ut som den vibrerade, och vad detta berodde på var oklart då andra bilar med samma servo och mikrodator inte hade samma problem.

Bild på PWM-signalen obelastad.

Med bara en månad till tävling och ingen riktig idé om vad som var fel, påbörjade vi felsökning och testade del för del för att försöka se vad det var som gjorde att servot betedde sig konstigt. Efter mycket letande kom vi fram till att det var Arduinot eller koden som krånglade. Vi mätte upp utsignalerna med ett oscilloskop på Arduinot och den visade sig ge en pwm-signal som inte var en fyrkantsvåg, utan snarare en blandning av en sinus- och en fyrkantsvåg, något som med största sannolikhet var ett stort problem. Samtidigt kunde vi på oscilloskopet se hur spänningen sjönk vid belastning från 5v till 1.8v. Detta berodde på att strömförsörjningen till servot var väldigt hög, och vid närmare undersökning visade det sig att servot drog närmre 0.7A, något som enligt databladet skulle vara 1A. Vi kunde även se en fördröjning på PWM-timingen, signalen var lite långsammare än beräknat, och det berodde med största sannolikhet på att AD-omvandlingen som görs vid varje avläsning för de tre sensorerna tar väldigt lång tid, något som ledde till att det blev en fördröjning i resterande program.

Bild på PWM signalen belastad.

Nu hade vi alltså identifierat tre olika problem. Även fast det bara var tre veckor kvar till tävling så beslutade gruppen att ännu en gång göra en ny kod, men den här gången valde vi att göra det enkelt för oss, gå tillbaka och programmera proceduellt.

Problemet med strömförsörjningen till servot var kvarstående, här gjorde vi en enkel lösning vilket var att mata servot med en separat strömförsörjning som inte gick

genom Arduinot. Detta var relativt enkelt att lösa med hjälp av en 6V spänningsregulator. Det som var lite osäkert med den lösningen var att batteriets spänning pendlade mellan 7.5V och 8.5V och spänningsregulatorn hade en drop out på 2V detta skulle leda till att servot kunde få 7.9V om batteriets spänning sjönk till den nivån. Men eftersom vårt batteri var relativt stort så skulle spänningen med största sannolikhet aldrig sjunka så lågt.

För det sista problemet med AD-omvandlingen bestämde vi oss för att använda två Arduinokort. Detta dels för att koden ändå skulle skrivas om, dels för möjligheten att först göra en kod för motorn och testa den, och sedan göra en kod för servot. På detta sätt behövde Arduinot för motorn endast göra en AD-omvandling för en komponent vilket var den främre sensorn. Medan Arduinot för servot nu behövde göra två AD-omvandlingar för de två sensorerna som läste av väggarna.

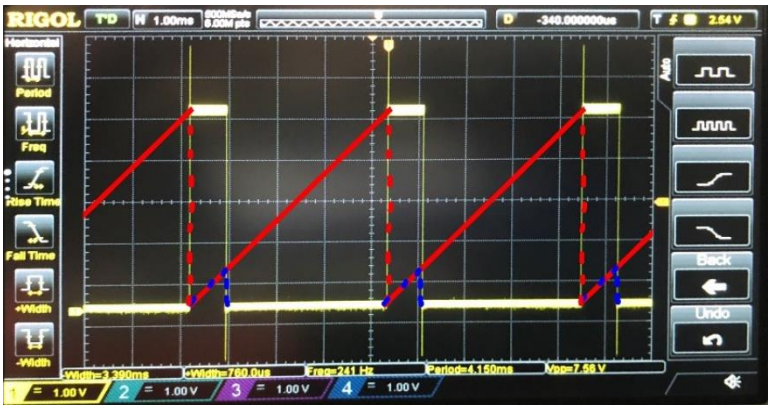
Vid det här stadiet hade vi fått tag på en till likadan bil som var tänkt fungera som reservbil, men i stället blev den uppgraderad till att agera andrabil då man enligt reglerna fick ställa upp med två bilar per grupp, alltså dubblades våra chanser att vinna tävlingen!

3.2 FINJUSTERING

För att få båda bilarna att fungera så skulle det nu monteras två Arduinos. Skrivas två nya koder till dem samt få strömförsörjningen till servot att fungera.
Det gick väldigt fort att få ordning på bil nr 1 då det gamla kortet kunde användas till styrning av endast motorn, och det som behövdes var en kod och kort till servot.
Till den andra bilen behövdes ett nytt kort med plats för två Arduinos och styrning för servo samt motor.

Den nya koden som konstruerades löste våra tidigare PWM-problem, i stället för att göra som i de första två varianterna, där först en ON-tid beräknades och sen togs denna tid minus en periodtid för att beräkna OFF-tiden, valde vi att använda två funktioner för att räkna ON- respektive OFF-tid.

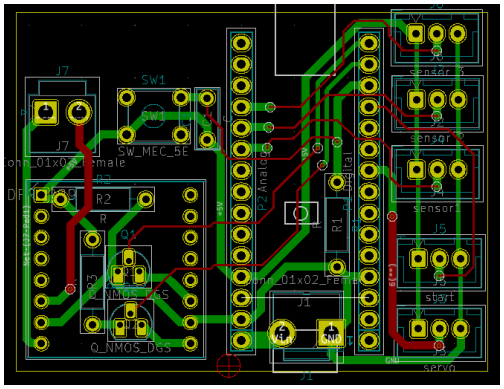
Nu användes en funktion för att räkna upp hela periodtiden, och vid start sattes motorn till ON.
En annan funktion hade som uppgift att beräkna ON tiden för att när den löpt ut sätta motorn till OFF.



Fungerande PWM där rött streck representerar funktionen som räknar periodtiden och blått representerar timern som beräknar ON tiden.

Att använda 4ms periodtid för både motor och servo var för att kunna använda samma kod för pwm-styrningen för båda Arduino så att inte två separata funktioner behövde skrivas då tiden var emot oss.

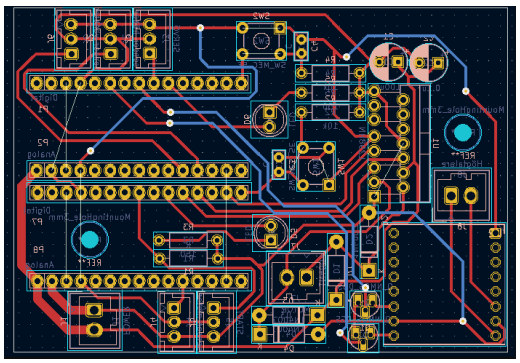
Tre nya kretskort designades, ett till den första bilen för att komplettera det första kortet, det nya kortet skulle styra servot, vi ritade också in en mp3-spelare som skulle spela upp motorljud, men det prioriterades bort.



Kort 1 schema i appendix

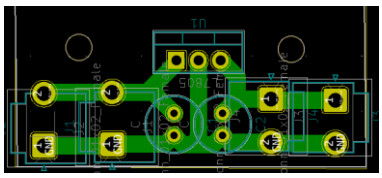
Ett till som skulle sitta på bil nummer två, detta fick bli ett kort där allt satt monterat på samma mönsterkort, och som vi valde att montera ovanpå bilen i stället för vid sidan för att testa en annan placering på sensorerna som istället för framför placerades vid sidan av bilen.

Det sista blev ett litet kort vars syfte var att leverera 6V och minst 1A till servot, detta gjordes med en 6V spänningsregulator L7806cv [5] med en kondensator kopplat parallellt över ingången och minus och en över utgången och minus

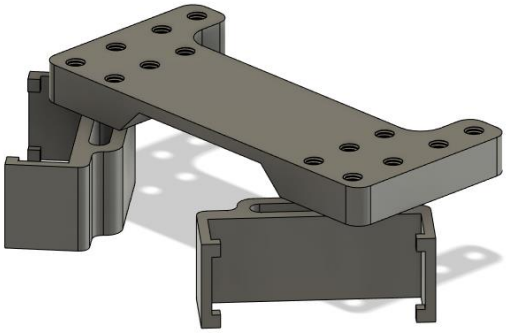


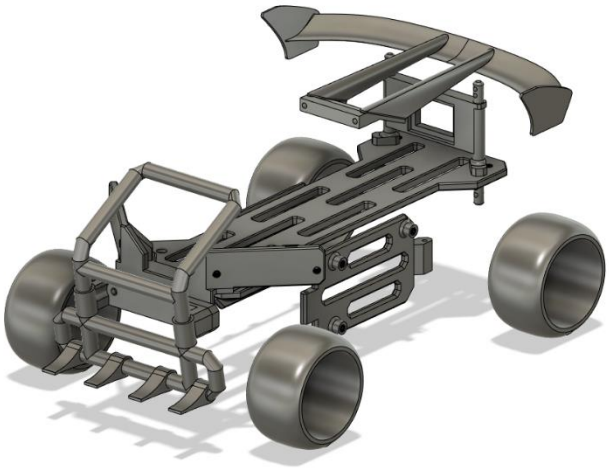
Kort 2 schema i appendix

Med dessa tre problem lösta kunde bil nr 1 med en vecka kvar till tävling testas med väldigt gott resultat. Bil nr 2 som i stort sett är identisk, det som skiljer är placering av komponenter och sensorer, men samma mjukvara och samma hårdvara, blev färdig dagen innan tävlingen

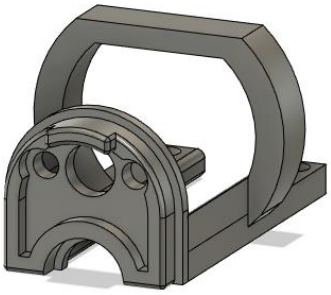


Kort 3 schema i appendix





Med den korta tiden kvar innan tävling blev det inte så mycket finjustering, en backfunktion lades till efter förtydligande av reglerna där poängavdrag inte delas ut om bilen kör åt fel håll, vilket var en nyhet för gruppen. Detta var dock inte helt enkelt då motorn som satt i bilen var snabb, men inte stark, den orkade med andra ord inte backa om den inte fick en hög spänning, och då fanns risken att dra sönder plastkuggarna på motorn om den skulle gå från full fart fram till full back, så en ny motorhållare designades till en motor som inte var lika snabb, men betydligt starkare då statorn på denna var kortare men betydligt bredare.

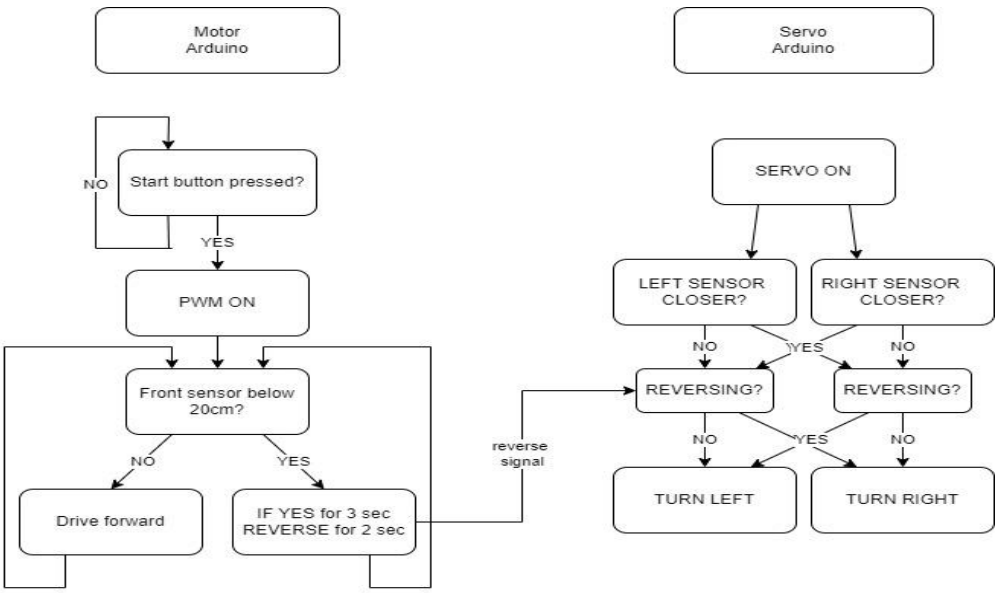


Ny motorhållare.

En signal från Arduinot som styrde motorn till Arduinot som styrde servot fick även skapas så att när backen läggs i så byts riktningen på styrningen för att kunna komma loss från hinder.

Med dessa funktioner och alla våra problem lösta kunde resterande tid läggas på att justera PID-regleringen som reglerade hur mycket bilen skulle svänga i förhållande till avståndet från väggarna.

PID-regleringen har som syfte att reglera styrningen där P (proportionella) är det faktiska felet, detta var den enda parameter som var tänkt initialt, att om ena sensorn mäter 40 cm till vägg medan andra mäter 80 cm, då är felet 50% och bilen ska då svänga 50% åt ena hållet. Till detta lades en I- och en D-del till, D-delen har som uppgift att skynda på korrigeringen av ett fel, så att den börjar svänga skarpt när ett fel uppstår medan I-delen har i uppgift att inte överstyra när bilen börjar hamna i mitten av vägen. Här valdes en PID-reglering som var ganska kvick i styrningen då en bil som konstant ligger i mitten av vägen sannolikt kommer ha problem med att alla andra bilar också ligger där. En reglering som i stället gjorde att bilen sicksackade hade troligtvis större chans att väja för andra bilar som stod i vägen, vilket verkade bättre och därför valdes.



4 RESULTAT

Slutresultatet blev två färdiga bilar med nästan alla funktioner som var tänkta från början, några blev bortprioriterade då viktigare funktioner implementerades. Den funktion som fick prioriteras bort var en mp3-spelare som skulle spela motorljud under tävlingen, den blev ersatt av en backfunktion som fick implementeras i sista minut, den var extra knepig eftersom det nu satt två mikrodatorer som inte kommunicerade med varandra. Lösningen blev effektiv även om den var primitiv, en 5V signal skickades från den mikrodator som styrde motorn så fort den började backa, och när den bytte riktningen igen så blev signalen 0V igen. En PID-reglering var inte heller tänkt från början men visade sig vara väldigt enkel att implementera och väldigt effektiv, dels då den hade väldigt bra justeringsmöjligheter, men framför allt för att den fungerade väldigt bra.

En annan sak som kom som en bonus var dubbla Arduino NANO, den valdes främst på grund av sin storlek men också för att den hade en klockfrekvens på 20MHz, detta visade sig dock inte stämma, utan den hade en frekvens på 16MHz precis som Arduino UNO, men eftersom vi i slutändan använde två mikrodatorer så blev det totalt 32MHz ihop, så bristen på MHz kompenserades upp, och vi hade nu dessutom en extra AD-omvandlare, så att två kunde användas i stället för bara en för sensorerna.

Ett annat mål som fick relativt hög prioritet var att bilarna skulle vara snyggare än en platta full med kablar, därför skulle allt byggas så att karosserna fortfarande kunde användas, på en liten bil var detta en utmaning, här var KiCad till väldigt stor hjälp men även tillgång till 3D skrivare, dessa två ihop gjorde att bilarna kunde byggas precis som vi ville.

5 TÄVLING

Vårt mål var från början att ha en färdig bil, men vi var ganska övertygade att vi hade en bra bil, backen funkade väldigt bra och tog oss ur väldigt många knipor och under ett par race behövdes inte våra bilar räddas alls.

Bil nr 1 fick banrekord på antal varv under första heatet vilket kändes fantastiskt bra, 18 varv! Och vi såg senare båda bilarna i A-final där tre av bilarna slutade på exakt samma antal varv, så det blev ett nytt race mellan våra två bilar och en till vilket gav oss en enorm strategisk fördel vad gäller hjälp, om en bil krockat och fastnade kunde den andra köra vidare och samla på sig varv om inte motståndaren bad om hjälp, men då fick han minuspoäng och låg efter båda våra bilar. Alltså, med två väldigt bra bilar, en bra strategi, och en fantastisk backfunktion tog vi första och andra plats!

6 DISKUSSION

Målen med detta projekt var att få en ökad förståelse för olika komponenter, men också att få förståelse för hur det kan vara att jobba i grupp.
Våra individuella mål var främst ökad kunskap i programmering då det kändes som den största utmaningen för alla gruppmedlemmar. Vikten av planering slog ingen av oss till en början utan vi kände att vi ville få planeringsfasen avklarad, och börja bygga bilen. Här dök problemet med engagemang upp, för här insåg vi att vi bara kommer vara två personer i gruppen. En fördel med detta kunde förstås bli att vi är få personer och få viljor att ta hänsyn till, det blir sannolikt lättare att prioritera och planera på kort sikt, men det blir också otroligt mycket mer arbete och ansvar på oss som var kvar. Att jobba agilt var inte ett problem, men det gjordes förmodligen inte på "rätt" sätt, med bara två personer i gruppen så kunde vi jobba väldigt bra ihop med gemensamt satta mål inför varje lektion och varje helg för att nå delmålen vi satte i projektstarten.

Det största problemet med att endast vara två personer i gruppen kom dock med tiden, eller snarare bristen på timmar. Med en budget på 150h per person saknades 450h vilket blev påtagligt då vi fick lägga otroligt många timmar utanför skolan, det krävdes långt mer än "våra" 300 h för att kunna leverera en körklar bil till tävlingsdagen, men i och med allt vi lärt oss genom projektet tycker vi ändå att det var värt det.

Det största lärdomarna har varit programmering, mycket för att det blev väldigt många timmar där vi bara skrev kod, och med en hel del problem så blev det både felsökning och problemlösning vilket varit väldigt givande.

Vi hade önskat mer kunskap i planering och hur man jobbar effektivt enligt den agila modellen. Fler och tätare uppföljningar, särskilt i det tidiga skedet av projektet där vi fick veta att tre gruppmedlemmar inte skulle delta, hur ska vi ställa oss till det, vilka krav kan vi ha på våra gruppmedlemmar och vilken nivå ska vi lägga oss på när det saknas så otroligt många timmar till projektet?

Den fördel vi kände att vi hade mot flera andra grupper var tillgång till väldigt nyttiga resurser som 3D-skrivare, oscilloskop och ett stort skruvsortiment. Detta gjorde att nästan alla ändringar vi behövde göra kunde lösas väldigt snabbt, och problemet med beställningar som var försenade eller inte dök upp alls var i stort sett obefintligt. Små detaljer kunde vi tillverka själva i stället för att kompletteringsbeställa, så värdet av rätt resurser och verktyg har varit enormt!

I slutändan fick vi båda ut vad vi ville, mycket erfarenhet och kunskap i kodskrivande, men vi fick också upp ögonen för vikten av bra planering och uppföljning, och även om vi klarade oss väldigt bra då vi bara var två personer som lyckades samarbeta bra så hade det varit en väldigt bra erfarenhet för arbetslivet att göra projektet i en större grupp.

Referenser

[1] Atmel, ”ATmega328P Datablad,” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[2] sharp, ”sharp datablad,” [Online]. Available: https://www.pololu.com/file/0J85/gp2y0a21yk0f.pdf.

[3] sparkfun, ”sparkfun datablad,” [Online]. Available: https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf.

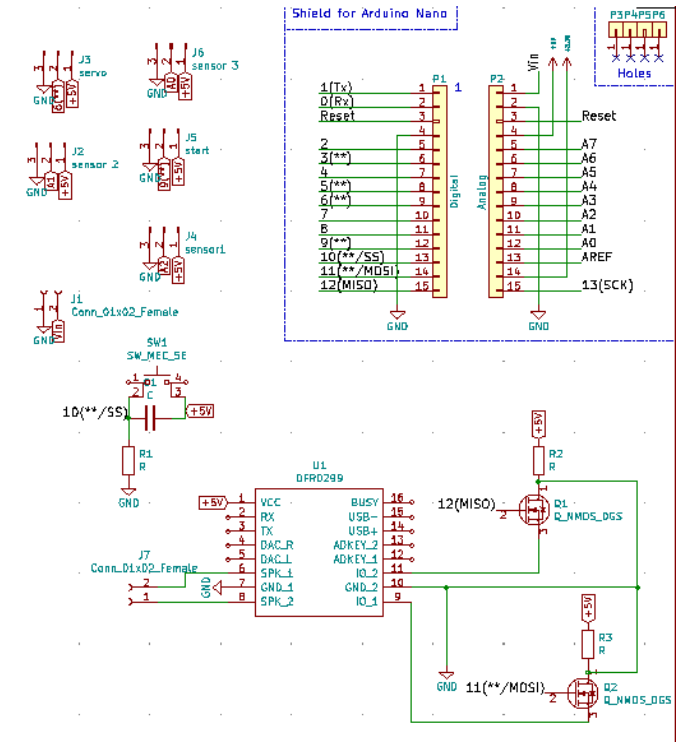
[4] autopartner, ”autopartner,” [Online]. Available: https://www.autopartner.se/elektronik/servon/savox/savox-sh-0257mg-servo-2-2kg-0-09s-alu-metalldrev-mikro.

[5] mouser, ”mouser datablad,” [Online]. Available: https://eu.mouser.com/datasheet/2/389/cd00000444-1795274.pdf.

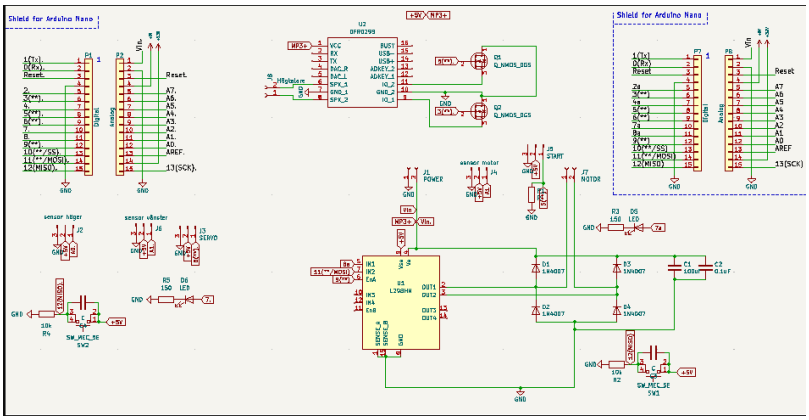
Verktyg som använts

	version	serienummer
KiCad	5.1.10	
Fusion 360		
Rigol DS1054		DS1ZA192309181
Fluke rms 115		47571218WS

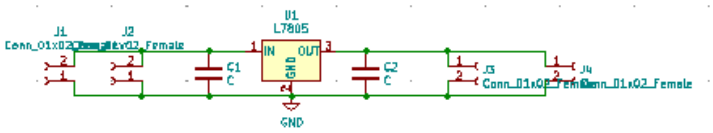
Appendix



Kort 1, tilläggs kort till bil nr1



Kort 2 till bil nr2



Kort 3 , strömförsörjningskort.

MOTORKOD

Main.c

```
#include "header.h"
/*****************************************************************************
 * I main anropas en setup funktion för alla initieringsrutiner och
*****************************************************************************/
int main(void)
{
  setup();

  while(true)
  {
      check_start_button();
  }
  return 0;
}
```

Header.h

```
#ifndef HEADER_H_
#define HEADER_H_

#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <stdbool.h>

#define BUTTON 1 //startknapp
#define BUTTON_IS_PRESSED (PINB & (1<<BUTTON)) //avläsning av startknapp

#define MOTOR 5
#define MOTOR_ON PORTD |= (1<<MOTOR)
#define MOTOR_OFF PORTD &= ~(1<<MOTOR)

#define CONNECTION 6 //koppling mellan båda arduinos för backfunktion
#define CONNECTION_ON PORTD |= (1<<CONNECTION)
#define CONNECTION_OFF PORTD &= ~(1<<CONNECTION)

#define MOTOR_DIRECTION1 7
#define MOTOR_DIRECTION2 0

#define SENSOR 1

#define MAX_DISTANCE 80.0
#define MIN_DISTANCE 50.0

#define ADC_MAX 1023
#define TOTAL_INTERRUPTS 500
#define PERIOD 4
#define INTERRUPT_TIME 0.000125

#define TOTAL_REVERSE_INTERRUPTS 400 //4.8s
#define INTERRUPTS_REQUIRED_FOR_REVERSE 150 //2.4s
#define REVERSE_INTERRUPT_TIME 0.016f

void setup();
uint32_t ADC_read();
uint16_t Calculate_distance();
```

```
void timer_on();
void timer_disable();
bool timer_elapsed();
void reverse_timer_on();
void reverse_timer_off();
bool reverse_timer_elapsed();
bool start_reversing();
bool duty_cycle_elapsed();
void get_new_duty_cycle();
void reverse_timer_on();
void reverse_timer_off();

void motor_toggle();
void motor_forward();
void motor_backwards();
void motor_enable();
void motor_disable();

void switch_pwm_mode();
void check_start_button();

void serial_print(const char* s); // Funktion för seriell överföring.
void serial_print_int(const char* s, const int number);

bool motor_enabled;
bool timer_enabled;
bool reverse_timer_enabled;
bool motor_reverse;
typedef enum {ON = 1, OFF = 0} period;

#endif /* HEADER_H_ */
```

Interrupts.c

```
#include "header.h"
/*************************************************************************
 * PCINT0_vect läser av om startknappen trycks ner, då sätts riktningen till
 * framåt och motorn togglas, funktionen användes vid tillfällig startknapp.
*************************************************************************/
ISR(PCINT0_vect)
{
    if(BUTTON_IS_PRESSED)
    {
        motor_forward();
        motor_toggle();
    }
    return;
}
/*************************************************************************
 * Timer 1 styr PWM funktionen för motorn, där timer elapsed räknar
 * periodtiden och dutycycle elapsed beräknar ON tiden för att stänga av
 * motorn när den löpt ut.
*************************************************************************/
ISR (TIMER1_COMPA_vect)
{
    if(timer_elapsed())
    {
        MOTOR_ON;
    }
    if(duty_cycle_elapsed())
    {
        MOTOR_OFF;
    }
    return;
}
/*************************************************************************
 * Timer 2 styr backfunktionen där reverse timer elapsed räknar upp till
 * 6 sekunder, om timern fortfarande är igång efter 3 sekunder börjar
 * bilen backa via funktionen start reversing. om avståndet överstiger
 * MAX_DISTANCE eller om timern löper ut så börjar bilen köra framåt igen.
*************************************************************************/
ISR (TIMER2_COMPA_vect)
{
    if(reverse_timer_elapsed())
    {
        motor_forward();
    }
    if(start_reversing())
    {
        motor_backwards();
    }
    return;}
```

ADC.c

```
#include "header.h"
/*********************************************************************
 * ADC-read läser av främre sensorn på bilen och återger ett värde mellan
 * 0-5v som omvandlas till ett digitalt värde mellan 0-1023.
*********************************************************************/
uint32_t ADC_read()
{
    ADMUX = ((1 << REFS0) | SENSOR);
    ADCSRA = ((1 << ADEN) | (1 << ADSC) | (1 << ADPS0) | (1 << ADPS1) | (1 <<
ADPS2));
    while ((ADCSRA & (1 << ADIF)) == 0) ;
    ADCSRA = (1 << ADIF);
    return ADC;
}

/*********************************************************************
 * calculate distance omvandlar värdet från ADC read och med en beräkning
 * lämnar ett värde mellan 10 och 80cm, en if sats läser av ifall avståndet
 * är under 20cm, då startar en timer som om den löper ut efter 3 sekunder
 * gör att bilen börjar backa.
 * En annan IF sats läser av ifall värdet understiger eller överstiger
 * bestämnda parametrar, är värdet över 65 ges värdet MAX_DISTANCE vilket
 * är 80, om det understiger MIN_DISTANCE vilket är 50 begränsas det till
 * 50 för att inte bilen ska köra långsammare än önskat.
 * Detta multipliceras sedan med 7 för att kunna användas med PWM timern
*********************************************************************/
uint16_t Calculate_distance()
{
    float in_signal = ADC_read() * 0.0048828125;
    double distance_in_cm = 29.988*(pow(in_signal, -1.173));
    if(distance_in_cm <= 20)
    {
        if(!reverse_timer_enabled)
        {
        reverse_timer_on();
        }
    }
    else
    {
        if(reverse_timer_enabled)
        {
            reverse_timer_off();
        }
        if(motor_reverse)
        {
            motor_forward();
        }
    }
```

```
    if(distance_in_cm >= 65)
    {
        distance_in_cm = MAX_DISTANCE;
    }

    if(distance_in_cm <= MIN_DISTANCE)
    {
        distance_in_cm = MIN_DISTANCE;
    }
    uint16_t on_time_interrupts = distance_in_cm * 7;
    return on_time_interrupts;
}
```

Motor.c

```
#include "header.h"

/*****************************************************************************
 * motor toggle läser av om motorn är enabled eller inte och togglar motorn
 * till motsatta värde.
*****************************************************************************/
void motor_toggle()
{
    if(motor_enabled)
        motor_disable();
    else
        motor_enable();
    return;
}

/*****************************************************************************
 * motor forward/backward sätter 2 siognaler till H bryggan som beroende
 * på vilken som är hög och låg gör att motorn backar eller kör framåt.
*****************************************************************************/
void motor_forward()
{
    motor_reverse = false;
    CONNECTION_OFF;
    serial_print("connection off");
    PORTD |= (1<<MOTOR_DIRECTION1);
    PORTB &= ~(1<<MOTOR_DIRECTION2);
    return;
}
```

```
void motor_backwards()
{
    motor_reverse = true;
    CONNECTION_ON;
    serial_print("connection on");
    PORTD &= ~(1<<MOTOR_DIRECTION1);
    PORTB |= (1<<MOTOR_DIRECTION2);
    return;
}

/*************************************************************************
 * motor enable/disable sätter motorn enable till true eller false, vid
 * enable så sätts motor enable till true och timer 1 startar.
 * vid disable så stängs timer 1 av och motor enable sätts till false
 * och motorn stängs av.
*************************************************************************/
void motor_enable()
{
    motor_enabled = true;
    timer_on();
    return;
}

void motor_disable()
{
    motor_enabled = false;
    timer_disable();
    MOTOR_OFF;
    return;
}

/*************************************************************************
 * check start button läser av om startknappen vid tävling är aktiv och
 * håller då motor enable true tills signalen försvinner.
*************************************************************************/
void check_start_button()
{
    if(BUTTON_IS_PRESSED)
        {
            motor_enable();
        }
    else
        motor_disable();
    return;
}
```

Serial.c

```c
#include "header.h"

static void serial_write_byte(const char data);
/*****************************************************************************
 * Serial funktionen användes främst för felsökning för att kunna läsa
 * av vad som hände under programmets gång, vid tävling så togs all
 * utskrift bort.
*****************************************************************************/
void serial_print(const char* s)
{
    for (register size_t i = 0; s[i]; i++)
    {
        serial_write_byte(s[i]);
        if (s[i] == '\n')
            serial_write_byte('\r');
    }
    return;
}

static void serial_write_byte(const char data)
{
    while ((UCSR0A & (1 << UDRE0)) == 0) ;
    UDR0 = data;
    return;
}

void serial_print_int(const char* s, const int number)
{
    char text[50];
    text[0] = '\0';
    sprintf(text, s, number);
    serial_print(text);
    return;
}
```

Setup.c

```
#include "header.h"

static void init_ports();
static void init_interrupts();
static void init_ADC();
static void init_timer();
static void init_serial();

void setup()
{
    init_ports();
    init_interrupts();
    init_ADC();
    init_timer();
    init_serial();
    motor_disable();
    motor_enabled = false;
    CONNECTION_OFF;
}
/*************************************************************************
 * init ports initierar startknappen (BUTTON), motorn (MOTOR),
 * motor riktningen (MOTOR_DIRECTION1-2) och anslutningen mellan de båda
 * korten för backfunktionen (CONNECTION)
*************************************************************************/
static void init_ports()
{
    DDRB |= (1<<MOTOR_DIRECTION2);
    DDRD |= ((1<<MOTOR) | (1 << MOTOR_DIRECTION1) | (1<<CONNECTION));
    PORTB |= (1<<BUTTON);
    return;
}

/*************************************************************************
 * init interrupts initierar interrupts på port B och port D
*************************************************************************/
static void init_interrupts()
{
    asm("SEI");
    PCICR |= ((1<<PCIE0) | (1 << PCIE2));
    PCMSK0 |= (1 << BUTTON);
    return;
}
```

```
/*************************************************************************
 * en initiering av ADC görs för annars kan första värdet bli dåligt.
*************************************************************************/
static void init_ADC()
{
    ADMUX = (1 << REFS0);
    ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADPS0) | (1 << ADPS1);
    while ((ADCSRA & (1 << ADIF)) == 0);
    ADCSRA = (1 << ADIF);
    return;
}

/*************************************************************************
 * timer 1 och 2 initieras i CTC mode där timer 1 styr PWM funktionen
 * och timer 2 styr backfunktionen, båda sätts till att räkna upp till 250,
 * men med olika prescalers
*************************************************************************/
static void init_timer()
{
    asm("sei");
    TCCR1B = ((0 << CS12) | (1 << CS11) | (0 << CS10) | (1 << WGM12));
    TIMSK1 = (1 << OCIE1A);
    OCR1A = 250;

    TCCR2B = ((1 << CS22) | (1 << CS21) | (1 << CS20));
    TCCR2A = (1 << WGM21);
    OCR2A = 250;
    return;
}

/*************************************************************************
 * init serial initierar utskrift för felsökning
*************************************************************************/
static void init_serial()
{
    UCSR0B = (1 << TXEN0); //ENABLE_SERIAL_TRANSFER;
    UCSR0C = ((1 << UCSZ00) | (1 << UCSZ01));//SET_TRANSMISSION_SIZE;
    UBRR0 = 103;//SET_BAUD_RATE_9600_KBPS;
    return;
}
```

Timer.c

```c
#include "header.h"

static uint16_t required_interrupts = TOTAL_INTERRUPTS;
static uint16_t required_interrupts_on;
static volatile uint16_t executed_interrupts;
static volatile uint16_t reverse_executed_interrupts;
static uint16_t reverse_total_interrupts = TOTAL_REVERSE_INTERRUPTS;
static uint16_t required_interrupts_for_reverse =
INTERRUPTS_REQUIRED_FOR_REVERSE;

/*************************************************************************
 * timer on startar timer 1
*************************************************************************/
void timer_on()
{
    TCCR1B = (1 << CS10) | (1 << WGM12);
    timer_enabled = true;
    return;
}

/*************************************************************************
 * timer disable stänger av timer 1
*************************************************************************/
void timer_disable()
{
    TCCR1B = 0x00;
    timer_enabled = false;
    return;
}

/*************************************************************************
 * timer elapsed räknar period tiden och hämtar värdet grån AD omvandligen
 * via get new duty cycle. om executed interrupts överstiger
 * required interrupts så returneras true och executed interrupts nollställs
*************************************************************************/
bool timer_elapsed()
{
    if (++executed_interrupts >= required_interrupts)
    {
        executed_interrupts = 0x00;
        get_new_duty_cycle();
        return true;
    }
    return false;
}
```

```
/*************************************************************************
 * duty cycle elapsed räknar ON tiden, och returnerar true om
 * executed interrupts överstiger required interrupts, då stängs motorn av.
*************************************************************************/
bool duty_cycle_elapsed()
{
    if (executed_interrupts >= required_interrupts_on)
    {
        return true;
    }
    return false;
}

/*************************************************************************
 * get new duty cycle används för att hämta ett nytt värde från AD
omvandligen.
*************************************************************************/
void get_new_duty_cycle()
{
    required_interrupts_on = Calculate_distance();

    return;
}

/*************************************************************************
 * reverse timer on startar timer 2 för backfunktionen.
*************************************************************************/
void reverse_timer_on()
{
    TIMSK2 = (1 << OCIE2A);
    reverse_timer_enabled = true;
    return;
}
/*************************************************************************
 * reverse timer off stänger av timer 2 för backfunktionen och nollställer
 * räknaren.
*************************************************************************/
void reverse_timer_off()
{
    reverse_executed_interrupts = 0x00;
    TIMSK2 = 0x00;
    reverse_timer_enabled = false;
    return;
}
```

```
/*************************************************************************
 * reverse timer elapsed räknar antaled avbrott och stänger av timer 2
 * om timern löper ut
*************************************************************************/
bool reverse_timer_elapsed()
{

    if (++reverse_executed_interrupts >= reverse_total_interrupts)
    {
        reverse_executed_interrupts = 0x00;
        return true;
    }

    return false;
}

/*************************************************************************
 * start reversing läser av om timer 2 nått önskat antal interrupts för
 * att starta backfunktionen, när värdet är uppnått startar backen tills
 * timern löper ut.
*************************************************************************/
bool start_reversing()
{

    if (reverse_executed_interrupts == required_interrupts_for_reverse)
    {
        return true;
    }
    return false;
}
```

SERVOKOD

Main.c

```c
#include "header.h"

/*************************************************************************
 * I main anropas en setup funktion för alla initieringsrutiner och
 * avläsning av AD omvandlaren för PID-reglering görs kontinuerligt.
*************************************************************************/
int main(void)
{
  setup();
  while(true)
  {
    set_input();
    duty_cycle = regulate();

  }

  return 0;
}
```

Header.h

```c
#ifndef HEADER_H_
#define HEADER_H_

#include "PID_Controller.h"

#include <avr/io.h>
#include <math.h>
#include <util/delay.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <stdbool.h>

#define CONNECTION 1
#define CONNECTION_ACTIVE (PINB & (1 << 1))

#define SENSOR_RIGHT 2
#define SENSOR_LEFT 0

#define SERVO_PIN 6
#define SERVO_ON PORTD |= (1 << SERVO_PIN)
#define SERVO_OFF PORTD &= ~(1 << SERVO_PIN)

#define MAX_DISTANCE 80.0
#define MIN_DISTANCE 10.0
```

```
#define ADC_MAX 1023
#define PERIOD 4
#define INTERRUPT_TIME 0.008f

void setup();
uint32_t ADC_read(const uint8_t PIN);
uint16_t Calculate_distance(const uint8_t PIN);

void serial_print(const char* s);
void serial_print_int(const char* s, const int number);

void servo_enable();
void motor_disable();

void timer_on();
void timer_disable();
bool timer_elapsed();
bool duty_cycle_elapsed();

void PID_print();

bool servo_enabled;
bool timer_enabled;

#endif /* HEADER_H_ */
```

ADC.c

```
#include "header.h"

/*********************************************************************
 * ADC-read läser av de två sensorerna på sidan och återger ett värde mellan
 * 0-5v som omvandlas till ett digitalt värde mellan 0-1023.
*********************************************************************/
uint32_t ADC_read(const uint8_t PIN)
{
    ADMUX = ((1 << REFS0) | PIN);
    ADCSRA = ((1 << ADEN) | (1 << ADSC) | (1 << ADPS0) | (1 << ADPS1) | (1 <<
ADPS2));
    while ((ADCSRA & (1 << ADIF)) == 0) ;
    ADCSRA = (1 << ADIF);
    return ADC;
}
```

```
/*************************************************************************
 * calculate distance omvandlar värdet från ADC read och med en beräkning
 * lämnar ett värde mellan 10 och 80cm, detta multipliceras sedan med 13 för
 * att kunna användas med pwm timern.
*************************************************************************/
uint16_t Calculate_distance(const uint8_t PIN)
{
    float in_signal = ADC_read(PIN) * 0.0048828125;
    uint16_t distance_in_cm = 29.988*(pow(in_signal, -1.173));
    uint16_t on_time_interrupts = distance_in_cm * 13;
    return on_time_interrupts;
}
```

Interrupts.c

```
#include "header.h"

/*************************************************************************
 * Timer 1 styr PWM funktionen för servot, där timer elapsed räknar
 * periodtiden och dutycycle elapsed beräknar ON tiden för att styra
 * positionen för servot där 1.5ms är rakt fram, 0.7ms är 90 grader vänster
 * och 2.3ms är 90 grader höger. periodtiden är 4ms vilket ger en
 * frekvens på 250Hz
*************************************************************************/
ISR (TIMER1_COMPA_vect)
{
    if(timer_elapsed())
    {
        SERVO_ON;
    }
    if(duty_cycle_elapsed())
    {
        SERVO_OFF;
    }
    return;
}
```

PID_Controller.c

```c
#include "header.h"

static void check_output();
static void regulate_backwards();
static double output1 = 0;

void init_pid_controller()
{
    output = 0x00;
    actual_value = 0x00;
    last_error = 0x00;
    intergral = 0x00;
    motor_direction = MOTOR_DIRECTION_FORWARDS;
}
/*************************************************************************
 * regulate regler värderna från set input för att genom en PID reglering
 * ge ett output som sen ska styra servot dör värdet från set input är 90
 * för rakt fram 0 för 90 grader vänster och 180 för 90 grader höger.
 * detta tas +80 för att få rätt antal interrupts för att matcha de tider
 * som servot behöver, dvs 0.7ms till 2.3ms, där 1.5 är rakt fram.
 * regulate backwards används om en signal från andra kortet kommer, då
 * sätts ett värde för att bilen ska byta riktning mot vad den gör om bilen
 * kör framåt.
*************************************************************************/
void regulate()
{

    if (CONNECTION_ACTIVE)
    {
        regulate_backwards();
    }
    else
    {
        double error = TARGET - actual_value;
        intergral +=  error;
        double derivative = error - last_error;
        output1 = TARGET + Kp * error + Ki * intergral + Kd * derivative;
        check_output();

        if(intergral >= 2000) intergral = 2000;
        if (intergral <= -2000) intergral = -2000;
        last_error = error;
        output = output1 + 80;
    if (output >= 250) output = 250;
    else if (output <= 90) output = 90;
    }
    return;
}
```

```
/*************************************************************************
 * check output har som syfte att begränsa väärden utanför de önskade
 * värderna från set input.
*************************************************************************/
static void check_output()
{
    if(output1 >= OUTPUT_MAX)
        output1 = OUTPUT_MAX;
    else if(output1 <= OUTPUT_MIN)
        output1 = OUTPUT_MIN;
    return;
}

static void regulate_backwards()
{
   double error = TARGET - actual_value;
    intergral +=  error;
    double derivative = error - last_error;
     output1 = TARGET + Kp * error + Ki * intergral + Kd * derivative;
    check_output();

    if(intergral >= 2000) intergral = 2000;
    if (intergral <= -2000) intergral = -2000;
    last_error = error;
    output = output1 + 80;  // centrum 170
    if (output >= 250) output = 250;
    else if (output <= 90) output = 90;
    if (output <= 170)
    {
        output = 230;
    }
    else
    {
        output = 90;
    }
}
```

PID_Controller.h

```
#ifndef PID_CONTROLLER_H_
#define PID_CONTROLLER_H_

#include <stdio.h>

#define SENSOR_MAX 1023.0
#define SENSOR_MIN 0.0

#define OUTPUT_MIN 0
#define OUTPUT_MAX 180
#define TARGET 90
#define Kp 0.7
#define Ki 0.00001
#define Kd 0.6

typedef enum { MOTOR_DIRECTION_FORWARDS, MOTOR_DIRECTION_BACKWARDS }
motor_direction_t;

double output;
double actual_value;
double duty_cycle;
double last_error;
double intergral;
motor_direction_t motor_direction;

void regulate();
void set_input();
void init_pid_controller();

#endif /* PID_CONTROLLER_H_ */
```

Serial.c

```c
#include "header.h"

static void serial_write_byte(const char data);
/*****************************************************************************
 * Serial funktionen användes främst för felsökning för att kunna läsa
 * av vad som hände under programmets gång, vid tävling så togs all
 * utskrift bort.
*****************************************************************************/
void serial_print(const char* s)
{
    for (register size_t i = 0; s[i]; i++)
    {
        serial_write_byte(s[i]);
        if (s[i] == '\n')
            serial_write_byte('\r');
    }

    return;
}

static void serial_write_byte(const char data)
{
    while ((UCSR0A & (1 << UDRE0)) == 0) ;
    UDR0 = data;
    return;
}

void serial_print_int(const char* s, const int number)
{
    char text[50];
    text[0] = '\0';
    sprintf(text, s, number);
    serial_print(text);
    return;
}
```

Servo.c

```c
#include "header.h"
/*************************************************************************
 * servo enable startar timmer 1 för pwm funktionen.
*************************************************************************/
void servo_enable()
{
    servo_enabled = true;
    timer_on();
    return;
}

/*************************************************************************
 * servo enable startar timmer 1 för pwm funktionen.
*************************************************************************/
void Servo_disable()
{
    servo_enabled = false;
    timer_disable();
    return;
}
```

Setup.c

```c
#include "header.h"
static void init_ports();
static void init_interrupts();
static void init_ADC();
static void init_timer();
static void init_serial();
void setup()
{
    init_ports();
    init_interrupts();
    init_ADC();
    init_timer();
    init_serial();
    servo_enable();
}
/*************************************************************************
 * init ports initierar servot och anslutningen mellan korten
*************************************************************************/
static void init_ports()
{
    DDRD |= (1<<SERVO_PIN);
    PORTB |= (1 << CONNECTION);
    return;
}
```

```
/*************************************************************************
 * init interrupts initierar interrupts på port B och port D
*************************************************************************/
static void init_interrupts()
{
    asm("SEI");
    PCICR |= ((1<<PCIE0) | (1 << PCIE2));
    return;
}

/*************************************************************************
 * en initiering av ADC görs för annars kan första värdet bli dåligt.
*************************************************************************/
static void init_ADC()
{
    ADMUX = (1 << REFS0);
    ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADPS0) | (1 << ADPS1);
    while ((ADCSRA & (1 << ADIF)) == 0);
    ADCSRA = (1 << ADIF);
    return;
}

/*************************************************************************
 * timer 1 oinitieras i CTC mode och räknar upp till 255.
*************************************************************************/
static void init_timer()
{
    asm("sei");
    TCCR1B = ((0 << CS12) | (1 << CS11) | (0 << CS10) | (1 << WGM12));
    TIMSK1 = (1 << OCIE1A);
    OCR1A = 255;
    return;
}

/*************************************************************************
 * init serial initierar utskrift för felsökning
*************************************************************************/
static void init_serial()
{
    UCSR0B = (1 << TXEN0); //ENABLE_SERIAL_TRANSFER;
    UCSR0C = ((1 << UCSZ00) | (1 << UCSZ01));//SET_TRANSMISSION_SIZE;
    UBRR0 = 103;//SET_BAUD_RATE_9600_KBPS;
    return;
}
```

Timer.c

```c
#include "header.h"

static volatile uint16_t required_interrupts = PERIOD / INTERRUPT_TIME;
static volatile uint16_t executed_interrupts;

/*****************************************************************************
 * timer on startar timer 1
*****************************************************************************/
void timer_on()
{
    TCCR1B = (1 << CS10) | (1 << WGM12);
    timer_enabled = true;
    serial_print("timer enabled\n");
    return;
}

/*****************************************************************************
 * timer disable stänger av timer 1
*****************************************************************************/
void timer_disable()
{
    serial_print("timer disabled\n");
    TCCR1B = 0x00;
    timer_enabled = false;
    return;
}
/*****************************************************************************
 * timer elapsed räknar period tiden för servot och nollställer räknaren
 * när executed interrupts är högre eller lika högt som
 * required interrupts.
*****************************************************************************/
bool timer_elapsed()
{
    if (++executed_interrupts >= required_interrupts)
    {
        executed_interrupts = 0x00;

        return true;
    }
    return false;
}
```

```
/*************************************************************************
 * duty cycle elapsed räknar ON tiden för servot som bestämmer position,
 * värdet jämförs med output som är utsignalen från pid regleringen.
*************************************************************************/
bool duty_cycle_elapsed()
{

    if (++executed_interrupts >= output)
    {
        return true;
    }
    return false;
}
```

TOF_Seonsor.c

```
#include "header.h"

static uint16_t left_sensor_input;
static uint16_t right_sensor_input;
static uint8_t mapped_left_sensor_input;
static uint8_t mapped_right_sensor_input;

void set_input(const double new_left_sensor_input, const double
new_right_sensor_input);
static double check_sensor_input(const double sensor_input);
static void TOF_map();
/*************************************************************************
 * set input läser av värderna från höger och vänster sensor för att
 * mappa dessa och ge ett värde motsvarande bilens placering för att
 * funktionen regulate sedan ska kunna bestämma hur starkt bilen ska
 * svänga åt ett håll för att kompensera för ev fel.
*************************************************************************/
void set_input(const double new_left_sensor_input, const double
new_right_sensor_input)
{
    uint16_t ADC_Left = Calculate_distance(SENSOR_LEFT);
    left_sensor_input = check_sensor_input(ADC_Left);
    uint16_t ADC_Right = Calculate_distance(SENSOR_RIGHT);
    right_sensor_input = check_sensor_input(ADC_Right);
    TOF_map();
    actual_value = TARGET + mapped_left_sensor_input -
mapped_right_sensor_input;
    return;
}
```

```
double check_sensor_input(const double sensor_input)
{
    if (sensor_input > SENSOR_MAX)
        return SENSOR_MAX;
    else if (sensor_input < SENSOR_MIN)
        return SENSOR_MIN;
    return sensor_input;
}

static void TOF_map()
{
    mapped_left_sensor_input = left_sensor_input / SENSOR_MAX * TARGET;
    mapped_right_sensor_input = right_sensor_input / SENSOR_MAX * TARGET;
    return;
}
```