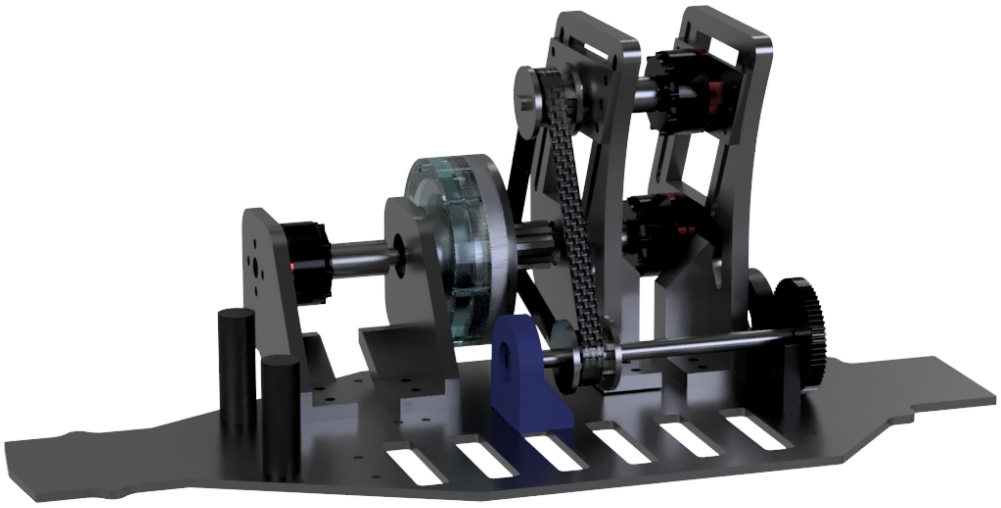
CHALMERS
UNIVERSITY OF TECHNOLOGY



Modellering och Konstruktion av Toyota Prius Hybriddrivlina

EENX15-18-14

Kandidatarbete

Alexander Andersson, Daniel Chai, Jane Jardebrand
Fredrik Lagerstedt, Fredrik Sveide, Martin Rattfelt

Institutionen för Elektroteknik
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sverige 2018

Kandidatarbete 2018:14

Modellering och Konstruktion av Toyota Prius Hybriddrivlina

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2018

Modellering och Konstruktion av Toyota Prius Hybriddrivlina
Alexander Andersson, Daniel Chai, Jane Jardebrand
Fredrik Lagerstedt, Fredrik Sveide, Martin Rattfelt

Handledare: Nikolce Murgovski
Examinator: Jonas Fredriksson

Kandidatarbete 2018:14
Institutionen för Elektroteknik

Chalmers Tekniska Högskola
SE-412 96 Göteborg

Omslag: Illustration av den slutgiltiga designen på den nedskalade modellen av Toyota Prius hybriddrivlina.

Skrivet i LaTeX
Göteborg, Sverige 2018

Modellering och Konstruktion av Toyota Prius Hybriddrivlina
Alexander Andersson, Daniel Chai, Jane Jardebrand
Fredrik Lagerstedt, Fredrik Sveide, Martin Rattfelt
Institutionen för Elektroteknik
Chalmers Tekniska Högskola

Abstract

Hybrid vehicles are very relevant the current market climate, fueled by rising petrol prices and environmentally conscious consumers. In order to gain a deeper insight into hybrid vehicles, this report will cover the production of a scaled-down powertrain similar to the Toyota Prius's. It should also be possible to intuitively understand which engine is active at any given moment by graphing the velocities of each motor on a computer. The procedure has been to study scientific texts within the fields of modelling and optimization. The result of the study illustrates the scaled-down model starting from a standstill and how the different motors behave during such a process. A safety feature drastically decreasing the performance of the engines was detected late in the development process, which limited the ability to gain simulated results. Nevertheless, a good foundation for future development opportunities have been created. A further development to investigate is the possibility of applying the method for minimizing fuel consumption on the the scaled down model.

Sammandrag

Hybriddrivna fordon är mycket aktuella i ett marknadsklimat färgat av stigande bensinpriser och miljömedvetna konsumenter. I syfte att få en djupare insikt i hybriddrivlinor kommer denna rapporten redovisa framställandet av en nedskalad drivlina likt den som sitter i Toyotas Prius. Det ska även vara möjligt att på ett intuitivt sätt förstå vilka motorer som arbetar genom illustration i ett grafiskt gränssnitt. Tillvägagångssättet har varit att studera vetenskapliga texter inom områden som optimering och modellering. I övrigt har en iterativ protypande process applicerats. Resultatet av studien illustrerar den nedskalade modellen med hybriddrift startandes från stillastående och hur de olika motorerna beter sig under en sådan process. En säkerhetsfunktion som begränsade prestandan hos motorerna upptäcktes sent i utvecklingsprocessen vilket begränsade möjligheterna att få simulerade resultat. Trots detta har en bra grund för framtida utvecklingsmöjligheter skapats. En vidareutveckling att undersöka är möjligheten att applicera den redovisade metoden för minimering av bränsleförbrukning på drivlinan på den uppställda modellen.

Keywords: Hybrid, Modellering, Optimering, Prius, HSD

Författarnas Tack

Författarna vill tacka sin handledare Nikolce Murgovski för sin ovärdeliga hjälp i arbetet. Personalen i prototyplabbet ska ha tack för sin hjälp under byggprocessen.

Alexander Andersson, Daniel Chai, Jane Jardebrand
Fredrik Lagerstedt, Fredrik Sveide, Martin Rattfelt
Göteborg, Maj 2018

Terminologi

Arduino Ett mikrokontrollerkort som kan skicka instruktioner till hårdvara, styra hårdvaran och skriva ut information till en dator.

Drivlina Systemet som skapar och förflyttar drivkraften i ett fordon till dess hjul.

Körcykel En standardiserad simulering av en bils hastighet över tid, kan även inkludera terränginformation.

Elhybrid Ett fordon vars primära förbränningsmotor kompletteras med en eller flera elmotorer.

Flyttal Ett numeriskt värde av typen *Float* i programmeringskod.

Innehåll

Figurer

1

Introduktion

Den traditionella förbränningsmotorn har varit förstahandsvalet för att driva de flesta väg-, vatten- och luftfarkoster under hela 1900-talet. Trots stora tekniska landvinningar kan förbränningsmotorns dagar vara räknade när den globala tillgången på olja börjar sina och den ökade miljömedvetenheten under 1900-talets senare hälft lett till ett sökande efter nya metoder för framdrift. Elbilen ses av många som en direkt ersättning till den traditionella bensinslukaren men trots en klart simplare motor relativt en modern förbränningsmotor sätter batteritekniken för närvarande begränsningar i räckvidd och höjer priserna på dessa fordon. Utöver detta behöver elbilsägare acceptera uppladdningstider om inte laddstationer används, vilket gör full eldrift olämplig i områden utan denna infrastruktur. Av dessa anledningar är det av intresse att undersöka hybriddrift för att finna en optimal blandning av de bästa egenskaperna hos båda lösningar. Genom att kombinera elmotorer med en förbränningsmotor kan låg bränsleförbrukning och god räckvidd åstadkommas och därmed kan den miljömässiga påverkan av fordonen minskas.

1.1 Bakgrund

Som tidigare nämnt är hybriddrivna fordon mycket aktuella i ett marknadsklimat färgat av ökande bensinpriser och miljömedvetna konsumenter. Trots att de kan tros vara ett nytt påfund har elbilar och hybrider rötter tillbaka till slutet av 1800-talet. Trots elfordons lovande start i bilismens tidiga skede fick förbränningsmotorer stort genomslag från och med 1910-talet, och med stor tillgång billig olja kunde de dominera marknaden totalt. [1] Den största delen av utvecklingen har dock skett på senare årtionden sedan slutet på 1990-talet då Honda och Toyota båda lanserade bränslesnåla hybridmodeller, Insight och Prius. Av dessa skulle Prius visa sig vara långlivad och bli den antagligen mest kända hybriden bland allmänheten, i skrivande stund är den fjärde generationen av Prius aktuell på marknaden.

Definitionen av en hybrid som ett ”fordon med flera olika motortyper” är brett nog för att tillåta ett antal manifestationer av hybridkonceptet att existera, något som uppmärksammas som förvirrande för allmänheten. [3] Detta då den underliggande

tekniken och konfigurationen av drivlinan är den avgörande faktorn för hur hybriderna kategoriseras.Hybrider delas upp i olika "klasser", Mild, Full och Plug-in. Milda hybrider har egenskapen att elmotorn kan agera som komplement till förbränningsmotorn men bilen kan ej köras på endast eldrift, i motsats till fullhybrider. Plug-in hybrider har möjlighet att laddas externt vilket ger dem möjlighet att köra mer på eldrift än de övriga typerna. Gemensamt för alla typer är dock möjligheterna att bromsa med elmotorn och på så sätt ”regenerera” en del av batteriet, dessutom kan förbränningsmotorn stängas av vid stopp och på så sätt undvika onödig tomgångskörning. [4]

Hybrider kan även delas upp som Parallell-, Serie- och Power-split hybrider vilka skiljer sig på hur drivlinan är konfigurerad. Parallella hybrider har både en förbränningsmotor och en elmotor kopplade till drivaxeln via en vanlig växellåda. I detta fall agerar elmotorn starthjälp, extra drivkraft och om det är en fullhybrid kan elmotorn driva fordonet på egen hand. Seriehybrider använder i sin tur elmotorn för framdrift men låter en annan motor eller energikälla ladda elsystemet, dessa kallas även för Extended Range Electric Vehicles (EREV). [5][1]

Power-split hybrider, även kallade serie-parallella hybrider, blandar båda konfigurationerna så att ett fordon kan drivas av bara förbränning, bara el eller båda kombinerat, detta är systemet som Toyota Prius använder. Konfigurationen i detta fordon består av en förbränningsmotor, en elmotor och en elgenerator för regenerativ bromsning, alla sammankopplade till drivaxeln via en växellåda. [7]

1.2 Syfte och mål

Syftet med arbetet är att framställa en skalad modell av en elhybriddrivlina monterat på ett litet bilchassi. Drivlinan består av flera motorer sammankopplade med en effektdelningsenhet och motorernas varvtal och energiåtgång skall kunna mätas separat ifrån varandra. Med denna data avses effektiviteten i hybriddrivlinan undersökas och lösningen optimeras. För detta syfte kommer en testbänk med ställbar last konstrueras, där målet är att dynamiskt kunna variera lasten på drivlinan så att en komplett körcykel kan simuleras. Målet är även att drivlinan skall kunna styras och eventuellt köras trådlöst för att visas upp på demonstrationer.

Slutprodukten ska vara en bil i en mindre skala som ska ha förmågan att kunna accelerera framåt. Det ska vara möjligt att mäta motorernas uteffekt och varvtal var för sig. Regenerativ uppladdning är tänkt att implementeras eller modelleras tillsammans med en möjlighet att fjärrstyra fordonet för demonstrationssyfte.

1.3 Problemformulering

Det huvudsakliga problemet som uppgiften behandlar är nedbrytandet, systematiseringen och konstruktionen av drivlinan. Problemet kan delas upp i några huvuddelar, först och främst valet av drivlina, modelleringen av denna, styrningen av motorerna och till sist verifikation och tester av konstruktionen. För att få en god verklighetsförankring baseras konstruktionen på en redan existerande lösning vars funktion och konstruktion efterliknas. En tydlig begränsning är hur enkel implementationen av den valda drivlinan är i mindre skala då drivlinor med förbränningsmotorer i regel behöver en växellåda och en koppling för att effektivt tillvarata effekten ifrån motorn.

En snabb studie av olika tillverkares hybriddrivlinor visade att flertalet olika lösningar existerar, men att ett antal är ”påbyggnationer” av redan existerande förbränningsdrivlinor. Till exempel är Volvos hybrider ”Twin Engine” i skrivande stund ett parallellt system där förbränningsdrivlinan driver framhjulen och en elmotor driver bakhjulen. [6] Detta leder till ökad bränsleeffektivitet hos fordonet då eldriften kan kopplas in vid behov men då uppgiften inte omfattar konstruktion av en ren förbränningsdrivlina är denna lösning inte aktuell.

Konstruktionen av drivlinan omfattar tillverkandet av en plattform på vilken drivlinan kan monteras, denna innefattar motor- och styrkortshållare samt plats för övrig elektronik. Sensorer kommer behövas för att mäta motorernas varvtal och effekt, samt göra denna data tillgänglig för de styrsystem som kan komma att behövas. För att behandla datan ifrån de separata sensorerna och motorerna i konstruktionen kommer en microkontroller behövas till drivlinan.

För att verifiera dess funktion och utföra simulationer på drivlinan kommer en testbänk tillverkas i vilken fordonet kan spännas fast och en konstlast appliceras på drivlinan. Denna kommer behöva motorer som kan kopplas till drivlinans drivaxel och på så sätt leverera ett dynamiskt motstånd motsvarande en förbestämd bilfärd. Dessutom är det av intresse att användare ska ha möjlighet att själva sätta värden på rotationshastigheterna och kunna avläsa denna hastighet och den levererade effekten hos varje motor. Av detta skäl kommer även ett grafisk gränssnitt programmeras som kan kommunicera med och presentera data ifrån de microkontroller som styr drivlinan. Även testbänken kommer behöva ett gränssnitt för att ge möjlighet att mata in simuleringsvärden för körcykler så som beskrivts tidigare.

1.4 Avgränsningar

Arbetet har begränsats till att studera den drivlina Toyota utvecklat till sina Prius modeller då den är en väl beprövad fullhybrid, serie-parallell drivlina med både elmotor, generator och förbränningsmotor vilket ger många möjligheter till olika simuleringssenarion. Dessutom till skillnad från många andra lösningar från konkurrenter finns det gott om information fritt tillgänglig om denna drivlina i alla dess olika generationer. För att ytterligare begränsa arbetet och göra det hanterbart utan tidigare erfarenheter av mekanisk design kommer följande begränsningar att tas i beaktning:

- I så stor utsträckning som möjligt används färdiga komponenter till byggandet. Styrkort, motorer, samt chassi köps in och modifieras för att lösa uppgiften.

- Bilen kommer förenklas så att telemetri-data inte hämtas när den är i radiostyrt läge utan endast vid stillastående körning på dynamometern. Detta då det blir enklare att hantera datamängden när datorer kan kopplas in direkt istället för att strömma datan kontinuerligt via bluetooth eller Wi-Fi.

- Drivlinan och speciellt planetväxeln kommer vara funktionellt identisk men icke-skalenlig mot den riktiga Prius versionen. Den tillåts även ha andra drivhjul då Prius framhjulsdrift kan vara mekaniskt svårlöst om styrning implementeras.

- Den högsta prioriteten är att drivlinan fungerar, att fordonet ska fungera som en riktig bil är önskvärt men ett sekundärt mål.

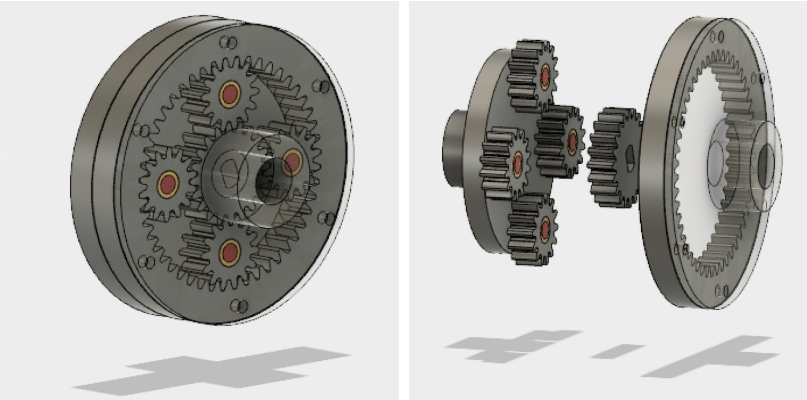
2

Teori

Detta kapitel introducerar drivlinans olika komponenter, deras samverkan och ger en introduktion i optimering av fordonet för en specifik körcykel. Del 2.1 förklarar vad som karakteriserar serie-parallella hybrider och hur en planetväxel fungerar. Del 2.2 presenterar hur minimering av Priusens bensinåtgång kan matematiskt modelleras samt lösas med Pontryagins minimeringsprincip.

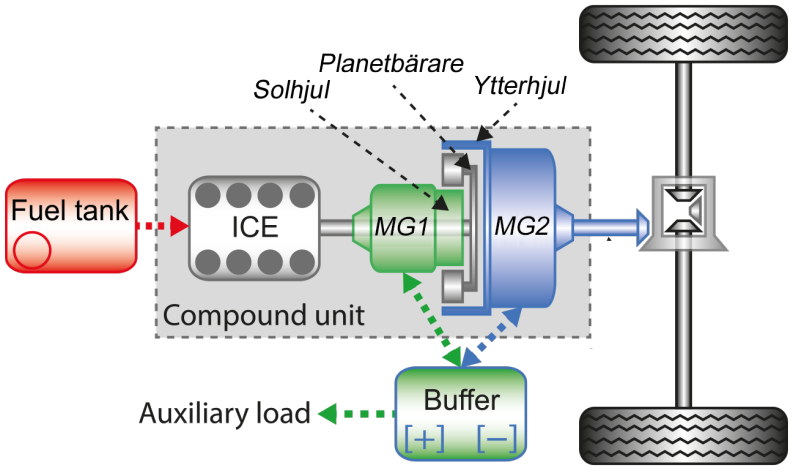
2.1 Serie-parallella hybrider och planetväxlar

Hybriddrivlinan modelleras på en Toyota Prius Hybrid Synergy Drive (HSD) drivlina, en ”serie-parallell hybrid”. I detta avsnitt kommer en redovisning av vad som karakteriserar dessa system. Som tidigare nämnt kan ett fordon med denna utformning drivas fram av endast elmotor eller förbränningsmotor men även av båda samtidigt. Toyotas hybriddrivlina har tre separata motorer som driver samma hjulaxel via en växellåda. I detta fall består systemet av en förbränningsmotor(ICE), och två elmotorer (MG1) och (MG2). Vars uppgift även är att agera som generatorer för att alstra ström till batteriet. För att uppnå effektiv hybriddrift måste motorerna kunna arbeta på olika varvtal och leverera olika vridmoment till drivlinan så att deras verkningsgrad är så hög som möjligt. För detta syftet har Toyota utvecklat en ”Power Split Device”, förkortat PSD, som sammankopplar de olika motorerna till utaxeln med en planetväxel. I figur 2.1 illustreras de tre huvudkomponenterna som tillsammans ger möjligheten att fördela kraften från motorerna. Figur 2.2 förklarar vilken motor som driver vilken del av växeln.



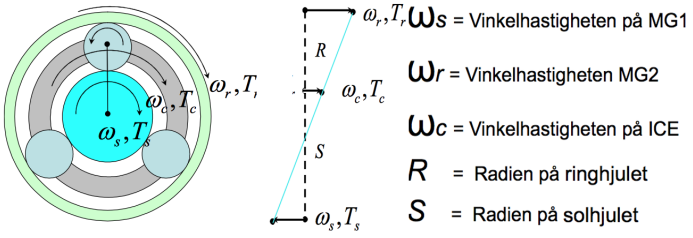
Figur 2.1: Den vänstra bilden illustrerar en planetväxel. Till höger visas planetväxels huvudkomponenter separerade från varandra. Från vänster: planetbärare med planethjul, solhjul och ytterring.

Då en förbränningsmotor inte kan starta från stillastående fungerar MG1 i första hand som en startmotor till ICE. MG1 fungerar även som en generator som tar tillvara på förbränningsmotorns överflödiga energi vid tex tomgång. Om drivlinan belastas under förbränningsmotorns optimala varvtal kan en del av kraften delas via planetväxeln (Power Split Device), och generera ström till batteriet via MG1 [8]. Alltså kan ICE arbeta i sitt optimala varvtal i ett större belastningsspann än vad som är möjligt i en konventionell förbränningsdrivlina.
MG2 hjälper ICE att driva fordonet genom att förbruka den lagrade energin i batteriet. Även MG2 fungerar som en generator i vissa fall [9]. När fordonet motorbromsar såväl som vid kraftigare inbromsningar så utövar MG2 motstånd på fordonets drivaxel, och lagrar den genererade energin i batteriet.



Figur 2.2: En illustration av villken motor som driver villken del av planetväxeln. ICE driver planetbäraren, MG1 driver solhjulet och MG2 driver ytterhjulet. Tagen ur ”Computationally efficient energy managementof a planetary gear hybrid electric vehicle” [11]

För att i ett senare stadie kunna dimensionera en planetväxeln måste först en modell av systemet beskrivas. Nedan redovisas hur de olika komponenternas vinkelhastighet i relation till varandra påverkar radien av respektive komponent.



Figur 2.3: Till vänster illustreras hur de olika komponenterna i en planetväxel roterar. Till höger redovisas ett hävarmsdiagram på relationerna mellan dessa komponenter. Tager ur "Configuration, Sizing and Control of Power-Split Hybrid Vehicles"[10]

Till höger i Figur 2.3 visas ett hävarmsdiagram som på ett intuitivt sätt illustrerar hur de olika vinkelhastigheterna och radierna ska uppfylla ett jämviktsläge. Detta jämviktsläge kan beskrivas enligt ekvation 2.1

$$\omega_s * S + \omega_r * R = \omega_c * (R + S). \tag{2.1}$$

Ur ekvation 2.1 kan följande formler för vinkelhastigheten av de olika delarna lösas ut enligt,

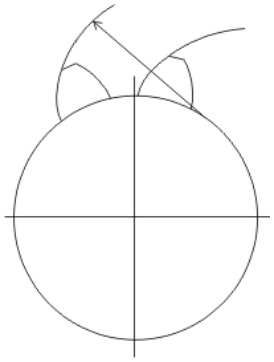
$$\omega_s = \frac{\omega_c * (R + S) - \omega_r * R}{S} \tag{2.2}$$

$$\omega_c = \frac{\omega_s * S + \omega_r * R}{R + S} \tag{2.3}$$

$$\omega_r = \frac{\omega_c * (R + S) - \omega_s * R}{S}. \tag{2.4}$$

2.1.1 Kugghjul

Kugghjul delas in i olika kategorier efter vinkeln på kuggarna gentemot kugghjulets axel, om kuggarna har en invändig eller utvändig placering, kuggarnas profil såväl som vilken modul kuggarna har.
Att kugghjulen är raka innebär att kuggarna är parallella med kugghjulets axel.
En evolventprofil fås genom att linda ett tänkt rep runt en cirkel, och sedan följa repets ände medans det lindas upp. Fördelen med detta är att två kugghjul i kontakt ständigt har en och endast en kontaktpunkt längs profilen, vilket tillåter kuggarna att rulla mot varandra. Detta ger en jämn och glappfri kraftöverföring, se figur 2.4.
Modulen är en standard för kugghjul, som fås genom att dela kugghjulets diameter från kuggarnas mitt med antalet kuggar.



Figur 2.4: En illustration på hur en evolventprofil tas fram. Den fås genom att följa ett tänkt snöres ände medans det lindas upp från en axel. Pilen är det tänkta snöret och kurvan som bildas blir den individuella kuggens profil.

2.2 Fordonsdynamik och optimering

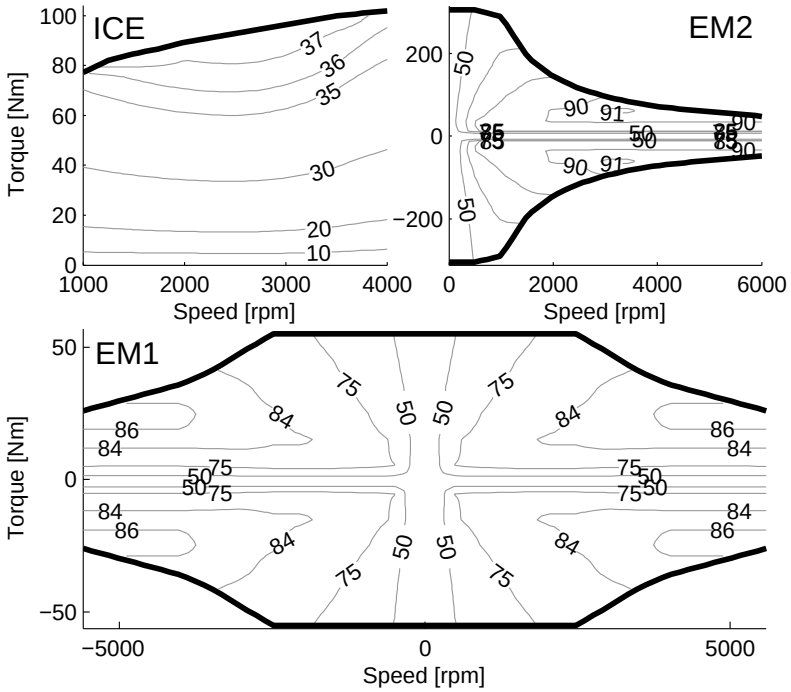
Fordonsdynamik omfattar de formler och ekvationer som tillsammans beskriver hur ett fordon beter sig under rörelse i termer av egenskaperna hos dess komponenter. Utifrån matematiska beskrivningar av motorn, växellådan, fjädringen m.m. kan hela fordonets beteende simuleras för varierande körcykler. För hybridbilar är det önskvärt att utföra simuleringar på körcykler och optimera för låg bränsleåtgång samt att batteriet avses ha samma laddnivå i början av färden som i slutet. Denna till synes motsägande avgränsning förenklar problemet då batteriet annars kan användas fritt när som helst tills det är urladdat men då fordonet, om det är en plug-in eller laddhybrid, inte kan säga ifall det kommer bli laddat i slutet av en färd är det säkrast att optimera för flera turer. På detta sett garanteras att fordonet inför varje ny körning har samma mängd batteri att ta ifrån och således kan utnyttja denna när det är som mest lämpligt.

I detta avseende presenteras nedan en förenklad lösning på detta problem för en Prius enligt metoden beskriven av ”Computationally efficient energy management of a planetary gear hybrid electric vehicle” [11].

Först introduceras de förbestämda tabellerna med värden på de förluster som uppstår i motorerna. Dessa förluster är direkt beroende av rotationshastigheten och det levererade vridmomentet ifrån varje motor. Beteckningarna E, 1 och 2 kommer hädanefter att användas för att särskilja bensinmotorn(ICE), samt de båda elmotorerna MG1 och MG2, beteckningen B syftar på effektförluster och P på effektvinster. Förlusterna beskrivs enligt,

$$\begin{aligned} B_E(\omega_E, T_E) \\ B_1(\omega_1, T_1) \\ B_2(\omega_2, T_2) \end{aligned} \tag{2.5}$$

och kan ses plottade i Figur 2.5 som effektivitetsområdena för varje motor.



Figur 2.5: De tre motorerns effektivitet plottad som funktion av deras levererade vridmoment och hastighet. Tagen ur "Computationally efficient energy management of a planetary gear hybrid electric vehicle" [11]

Utöver dessa är rotationshastigheterna och vridmomenten begränsade för varje motor, vilket avgränsar de områden inom vilka de kan arbeta, enligt

$$\begin{aligned} \omega_E &\in [\omega_{Eidle}, \omega_{EMax}] \\ \omega_1 &\in [\omega_{1idle}, \omega_{1Max}] \\ \omega_2 &\in [0, \omega_{2Max}] \end{aligned} \tag{2.6}$$

$$\begin{aligned} T_E &\in [0, T_{EMax}(\omega_E)] \\ T_1 &\in [T_{1min}(\omega_1), T_{1Max}(\omega_1)] \\ T_2 &\in [T_{2min}(\omega_2), T_{2Max}(\omega_2)]. \end{aligned}$$

Batteriet kan modelleras som en serie sammankopplade celler med en inre kapacitans som alla ger upphov till en terminalspänning och tomgångsspänning för batteriet. Terminalspänningen betecknas här u, C är kapacitans i Farad, n är antal celler, u_0 är tomgångsspänning för öppna kretsen vid minimala State of Charge (SoC) i volt.Detta ger den lagrade batterienergin som,

$$E_B = n\frac{C}{2}(u^2 - u_0^2) \tag{2.7}$$

där det i detta fall är värt att notera att n är givet av Priusbatteriets storlek och konfiguration. Utöver detta har batteriet även förluster på grund av inre resistanser vilket beror på den effekt P_B som dras ur batteriet. Förlusterna fås som,

$$B_B(E_B, P_B) = RC\frac{P_B^2}{2E_B + C_{u_0}^2 n}. \tag{2.8}$$

Effekten P_B är även begränsad av den upp- och urladdningsström i_{min} och i_{max} som batteriet kan hantera, alltså begränsas den enligt

$$P_B \in [i_{min}, i_{max}]\sqrt{n(\frac{C}{2}E_B + u_0^2 n)}. \tag{2.9}$$

För en given körcykel är bensinåtgången beroende av den rotationshastigheten och det levererade vridmomentet ifrån motorn samt dess förluster vid denna arbetspunkt enligt

$$P_f(\omega_E, T_E) = \omega_E T_E + B_E(\omega_E, T_E). \tag{2.10}$$

För att lösa bensinåtgången under hela färden löses integralen för denna ekvation mellan tiderna 0 och t_f då färden avslutas. Genom att sätta minimeringskrav på integralen

$$\min_{\omega_E, T_E, P_B} \int_0^{t_f} P_f(\omega_E, T_E)dt \tag{2.11}$$

kan bensinåtgången senare optimeras. Rotationshastigheten hos de tre motorerna förhåller sig enligt följande då de är sammankopplade genom planetväxeln. Här är r förhållandet mellan ytterringens- och solhjulets kuggar. På samma sätt förhåller sig även vridmomenten hos MG1 och bensinmotorn genom ett simpelt samband.

$$\omega_1 = (1 + r)\omega_E - r\omega_2 \tag{2.12}$$

$$T_E = -(1 + r)T_1 \tag{2.13}$$

Det totala vridmomentet och den totala rotationshastigheten som driver fordonet framåt blir summan av alla vridmoment och hastigheter enligt följande

$$\omega_E T_E + \omega_1 T_1 + \omega_2 T_2 = \omega_2(T_d + T_{Brm}). \tag{2.14}$$

Batterienergin påverkas således av hur mycket som dras ifrån det vid varje givet ögonblick samt förlusterna i varje elektriskt system. P_B påverkas av de övriga systemen enligt

$$P_B = \omega_1 T_1 + \omega_2 T_2 + B_1(\omega_1, T_1) + B_2(\omega_2, T_2) + B_B(E_B, P_B) + P_a \tag{2.15}$$

där P_a är alla system som inte är involverade i drivlinan så som stereo, AC, belysning etc. Batteriets lagrade energi är begränsad av dess totala kapacitet samt den lägsta nivån till vilken det är säkert att ladda ur batteriet enligt

$$E_B \in n[E_{Bmin}, E_{Bmax}]. \tag{2.16}$$

Bilens energibehov påverkar den lagrade energin enligt

$$\dot{E}_b = -P_B. \tag{2.17}$$

Kravet att batterinivån skall vara samma vid start som mål avgränsar vi genom

$$E_B(0) = E_B(t_f). \tag{2.18}$$

Övriga avgränsningar och förenklingar involverar att $T_{Brm} \geq 0$ och att ω_2, T_d, P_a, r, n är alla förbestämda.

För att beräkna T_d utifrån körförhållanden behövs fordonsdynamiken för bilen ställas upp. I detta fall tas endast longitudinella krafter i hänsyn då körcyklerna som används inte innehåller data om ev. kurvor. Genom att utgå ifrån Newtons andra lag,

$$F_{tot} = ma \tag{2.19}$$

kan samtliga krafter som påverkar bilen ställas upp.

Bilen påverkas av ett antal krafter i form av vindmotstånd, potentiell energivinst ifall höjdskillnader infinner sig, samt rullmotståndet mellan däcken och asfalten. Luftmoståndet är en funktion av bilens frontalarea A_c, luftmotståndskonstanten C_d, lufttätheten ρ och bilens hastighet v, totalt sett som

$$\frac{\rho_a}{2} A_c C_d v^2 \tag{2.20}$$

där väder och temperatur antas vara konstanta så att även lufttätheten är det. Rullmotståndet ger upphov till en kraft vinkelrät mot bilens gravitationskraft vilket beräknas enligt

$$mgC_r \cos\alpha. \tag{2.21}$$

På samma sätt bestäms även den potentiella energin och samtliga ekvationer kan nu kombineras till

$$ma = \frac{T_t}{r_{hjul}} - \frac{\rho_a}{2} A_c C_d v^2 - mgC_r \cos\alpha - mg\sin\alpha. \tag{2.22}$$

Ur denna ekvation kan sedan T_t bestämmas då acceleration och hastighet fås ur körcykeln för varje tidspunkt. För att optimera systemet kvarstår fortfarande att bestämma kontrollsignalerna till varje motor. Ett effektivt sätt att göra detta på är med Pontryagins minimeringsprincip, arbetsgången är beskriven av Desineni Subbaram Naidu i boken Optimal Control Systems, sidan 69. [12]

Pontryagins minimeringsprincip går ut på att minimera ett dynamiskt system i enlighet med begränsningar för varje diskret tillstånd. Alltså genom att ställa upp systemet likt ovan kan optimala värden för varje styrsignal erhållas för varje tidpunkt. I detta fall sätter antalet datavärden från körcykeln vi vill optimera för begränsningar för hur många tillstånd vi behöver beräkna. Innan optimering vidtar summeras problemet som

$$\min_{\omega_E, T_E, P_B} \int_0^{t_f} P_f(\omega_E, T_E) dt \tag{2.23}$$

som begränsas och påverkas av

$$\omega_1 = (1+r)\omega_E - r\omega_2 \tag{2.24}$$

$$T_E = -(1+r)T_1 \tag{2.25}$$

$$\omega_E T_E + \omega_1 T_1 + \omega_2 T_2 = \omega_2 (T_d + T_{Brm}). \tag{2.26}$$

$$P_B = \omega_1 T_1 + \omega_2 T_2 + B_1(\omega_1, T_1) + B_2(\omega_2, T_2) + B_B(E_B, P_B) + P_a \tag{2.27}$$

$$\dot{E}_b = -P_B. \tag{2.28}$$

$$E_B \in n[E_{Bmin}, E_{Bmax}]. \tag{2.29}$$

där $T_{Brm} \geq 0$ och ω_2, T_d, P_a, r, n är alla förbestämda som tidigare beskrivits.

Systemet vi vill minimera beskrivs med Hamiltonoperatören H vilket i detta fall blir beroende av bensinåtgången P_f och batterienergin P_b, vilka beskrevs ovan. Kontrollsignalerna för varje motor sätts till det vridmoment som krävs i varje tillstånd och samlas i styrvektorn u.

$$u = \begin{bmatrix} T_E \\ T_1 \\ T_2 \end{bmatrix} \tag{2.30}$$

$$H = P_f(u) + \lambda(-P_b(u)) \tag{2.31}$$

Här är λ en vikt som avgör hur mycket av energin som får komma ifrån batteriet för varje tillstånd i körcykeln. För att Pontryagins minimeringsprincip skall vara applicerbar utan att ge icke kontinuerliga signaler så måste E_b begränsas så att batteriet inte är fulladdat eller helt tomt.

$$E_{bmin} < E_b(0) < E_{bmax} \tag{2.32}$$

För att således bestämma λ måste följande begränsning göras:

$$\dot{\lambda} = -\frac{\partial H}{\partial E_b} = 0 \tag{2.33}$$

Vilket således gör att λ blir en konstant som bestäms genom upprepade simuleringar där batterinivån övervakas. Om det visar sig att batterinivån är för låg efter en körning justeras λ upp och är batterinivån för hög justeras den ner.

3

Metoder

I detta kapitel kommer de metoder och processer som lett fram till ett förverkligande av projektet redovisas. Del 3.1 fokuserar på att beskriva det mekaniska framställandet av den nedskalade Toyota Prius modellen. Den elektroniska biten redovisas i del 3.2 där systemet som mäter ström och varvtal redovisas. Del 3.3 beskriver likt del 3.1 en mekanisk framställning men handlar istället om testbänken. Del 3.4 beskriver processen av framställningen och den slutgiltiga produkten av det grafiska gränssnittet.

3.1 Design och konstruktion av drivlinan

Detta avsnitt beskriver det mekaniska framställandet av den nedskalade modellen. I detta gås det igenom hur val av chassi påverkade dimensioneringen av andra komponenter, samt hur planetväxeln skalades och tillverkades.

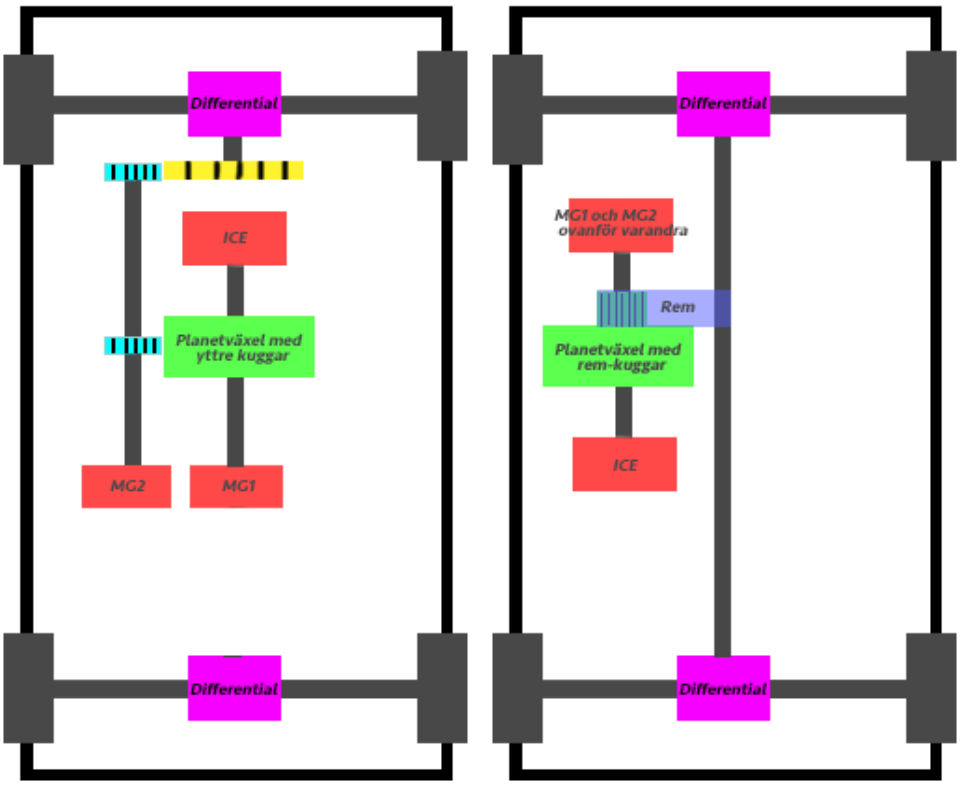
Figur 3.1: Modellnamn: ”RC modellbil Gatumodell 1:10 Reely Onroad-Chassis Elektrisk 4WD ARR”

Chassi

Drivlininan konstruerades på ett chassi med möjlighet att svänga och med inbyggd differentialväxel till hjulen. För att underlätta konstruktions och bygg -processen valdes det att köpa ett chassi med dessa egenskaper, detta lämnade mer utrymme för fokus på själva drivlinan. Chassit som valdes visas i figur 3.1. Det valdes pågrund av sitt låga pris vilket gjorde att storleken inte fick lika stor betydelse.

Designval av drivlina

Efter att studerat det inköpta chassit och med tidigare restriktioner rörande vilken motor som ska driva vilket kugghjul i planetväxeln kunde två layouter tas fram. De redovisas i Figur 3.2 nedan.



Figur 3.2: Två olika illustrationer av möjliga designval för applicering av hybriddrift till chassit. Till vänster visas alternativ 1 som använder kugghjul som kraftöverföring mellan planetväxeln och kardanaxeln, till höger visas alternativ 2 som istället använder en rem.

Alternativ 1 valdes bort då alternativet är känsligare för störningar. Att få en axel från MG2 att gå parallellt med både planetväxeln såväl som chassits ursprungliga drivaxel skulle innebära väldigt små toleranser. En rem med spännanording däremot är väldigt flexibel, vilket innebär att eventuella felaktigheter kan justeras i efterhand.

Dimensionering av planetväxeln

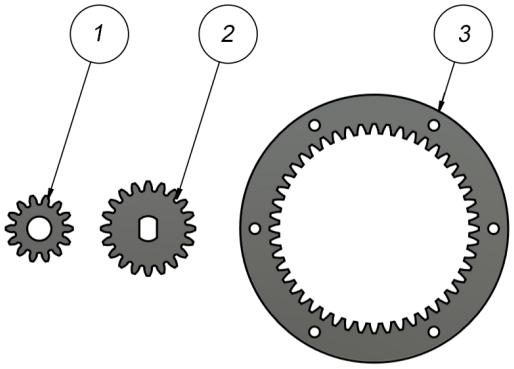
Målet med dimensioneringen av planetväxeln var att komma så nära den faktiska utväxlingen som möjligt. För att göra detta användes en modell som simulerar de olika motorernas varvtal [19]. Genom att undersöka ett visst simulerat motorscenario kunde motorernas varvtal allokeras. Dessa varvtal användes senare för att tillfredsställa följande formel, som lösts ut ur ekvation 2.1,

$$\frac{S}{R} = \frac{\omega_c - \omega_r}{\omega_s - \omega_c}. \tag{3.1}$$

Det uträknade resultatet blev att $S * 2.60105 = R$ vilket är ett ojämnt förhållande. Då det är omöjligt att ha en halv kugg drogs slutsatsen att förhållandet var felaktigt. Efter ytterligare efterforskningsarbete hittades den faktiska relationen som $S * 2.6 = R$. Den uträknade relationen är mycket nära den faktiska vilket leder till slutsatsen att simuleringens varvtal inte riktigt stämmer överens med verkligheten. Med den kunskapen bestämdes att använda det faktiska förhållandet.

Nästa steg blev att bestämma storleken på själva växeln. Efter närmare studier av det inköpta chassit kunde ringens ytterdiameter bestämmas till 64 mm för att passa utan att vidröra drivaxeln och inte sticka ut utanför chassit för mycket. Det valdes sedan att använda kuggmodul 1 då verktyg för att tillverka sådana kugghjul finns tillgängligt i prototypverkstaden. För att lämna plats åt sammansättning av komponenterna valdes innerdiametern till 52 mm. Eftersom kuggmodul 1 som användes blev diametern den samma som antalet kuggar. Detta ledde till följande antal kuggar på respektive kugghjul, ytterring: 52, planethjul: 15 och solhjul: 20. Men efter att provat detta förhållande i CAD visade det sig att varje kugghjul måste ha ett jämnt antal kuggar. Det bestämdes då att ta bort 4 kuggar från ytterringen vilket också innebar en minskning av innerdiametern på 4 mm. Det slutgiltiga antalet kuggar på respektive kugghjul fastställdes till, ytterring: 48, planethjul: 14 och solhjul: 20.

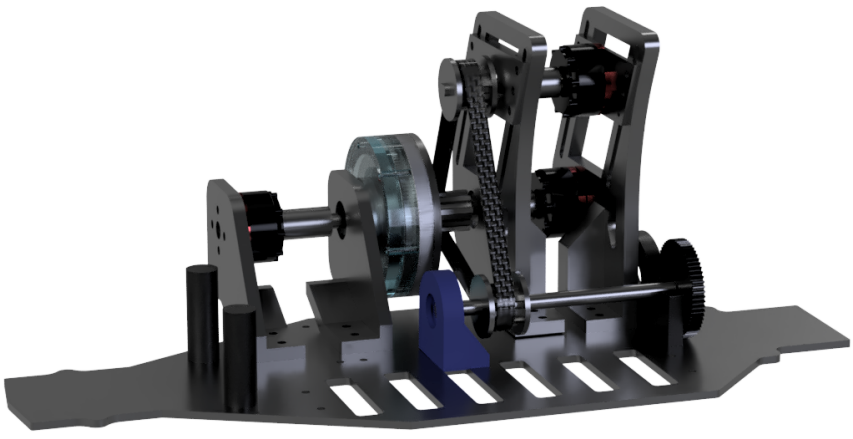
Nr	Komponent	Antal kuggar
1	Planethjul	14
2	Solhjul	20
3	Ytterhjul	48



Figur 3.3: Den slutgiltiga dimensineringen av planetväxeln. Där planethjulet har 14 kuggar, solhjulet 20 och ringhjulet har 48 kuggar.

Slutdesign och delar

Den slutgiltiga designen redovisas i figur 3.4 nedan. För att förenkla beskrivningen av design och tillverknings -processen kommer systemet brytas ned i mindre delar enligt följande. *Kraftöverföring*, *Hållare och motorer* samt *Planetväxeln*.

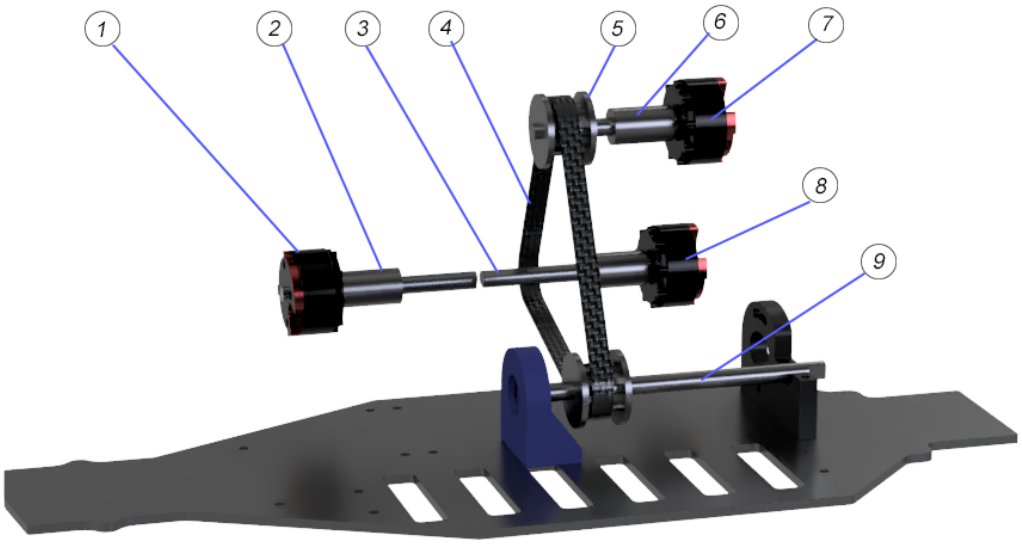


Figur 3.4: Illustration av den slutgiltiga designen av drivlinan.

Kraftöverföring

I figur 3.5 visas en bild av drivlinan där endast de komponenter som ansvarar för kraftöverföringen är synliga.

Nr	Komponent	Kommentar
1	Motor	ICE
2	Axel	Kraftöverföring ICE → planetbärare
3	Axel	Kraftöverföring MG1 → solhjul
4	Drivrem	Kraftöverföring från ringhjul och MG2 → drivaxel
5	Remskiva	Fäst med en stoppskruv mot komponent 6
6	Axel	Kraftöverföring MG2 → Ringhjul och drivaxel
7	Motor	MG2
8	Motor	MG1
9	Drivaxel	Kraftöverföring drivaxel → MG2 och ringhjul.

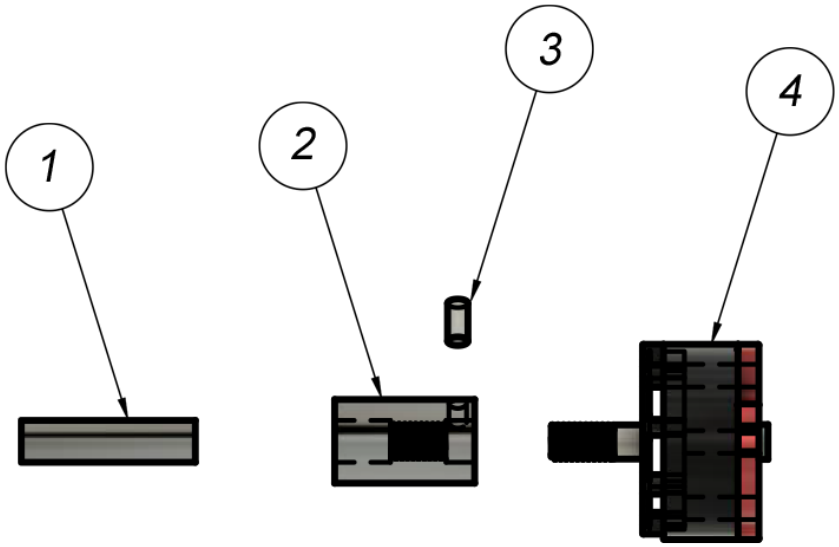


Figur 3.5: Illustration av drivlinan där endast kraftöverföringen är synlig.

Den största utmaningen var att på ett platseffektivt och icke-permanent sätt fästa axlarna till drivaxlarna på motorerna. Då de inköpta motorenas drivaxel bestod av icke utbytbara gängade axlar. Detta ledde till att en egen komponent var tvungen att framställs. Nedan illustreras en bild där det syns hur detta fäste är uppbyggt och hur det monteras på motorn.

När olika komponenter skulle fästas på axlarna användes det stoppskruvar som skruvades in i en liten fasning på respektive axel.

Nr	Komponent	Kommentar
1	Axel	Valfri axel
2	Invändigt gängad hylsa	Den del som kopplar ihop axeln med motorn.
3	Stoppskruv	Förhindrar hylsan från att skruvas av från motorn.
4	Motor	Valfri motor

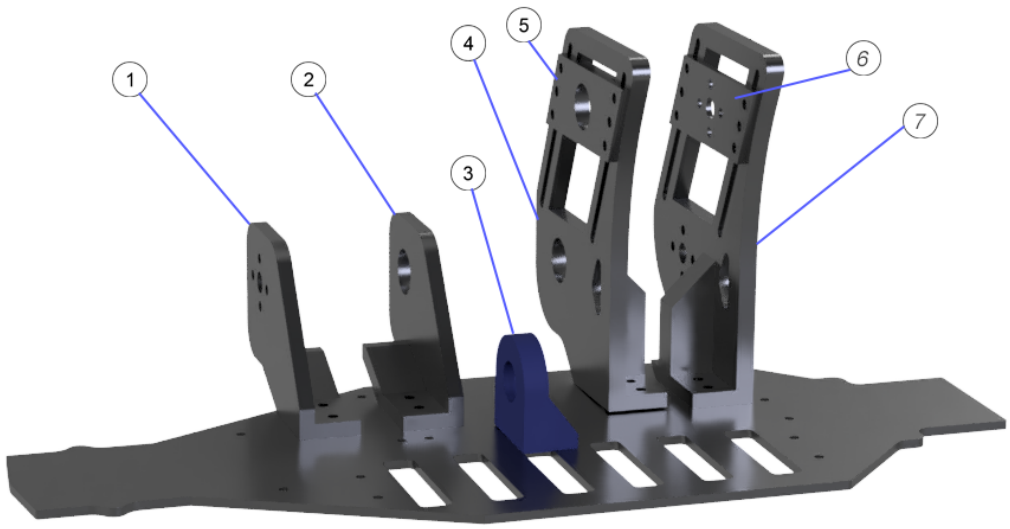


Figur 3.6: En illustration av hur axlarna monteras på motorn. Del nr:1 och 2 är ihoplimmade med ett poxylim. Del nr:3 skruvas igenom nr:2 och förhindrar på så vis rörelse. Del nr:2 och 4 skruvas ihop.

Hållare och fästanordningar

I figur 3.7 visas en bild av drivlinan där endast upphängningen av komponenterna är synliga. För att på ett enklare sätt beskriva designen delas illustrationen upp i två plan. Plan 1 är hålen som som går i linje med hålen från nr:1. Plan 2 är hålen som tillhör nr:5 och nr:6.

Nr	Komponent	Kommentar
1	Motorinfästning	ICE
2	Kulagerhållare	Skyddar MG1 från böjkrafter
3	Kulagerhållare	Håller drivaxeln på sin plats
4	Kulagerhållare	Skyddar MG1 från böjkrafter
5	Justerbar kulagerhållare	Skyddar MG2 från böjkrafter
6	Justerbar motorinfästning	MG2
7	Motorinfästning	MG1



Figur 3.7: Illustration av drivlinan där endast upphängningen av komponenterna är synliga.

Planetväxlar är mycket känsliga för störningar då det är kritiskt att kuggar som rullar mot varandra måste ha ett visst avstånd ifrån varandra. Då den framtagna planetväxeln är relativt liten är avståndet ifråga mycket litet. Detta betyder att hålen som tillhör vardera plan måste gå precis i linje med varandra. Detta resulterade i att delarna valdes att maskinframställas. Metoden som valdes var 3D-skrivare tack vare dess flexibilitet att snabbt skriva ut nya designer.

Komponenterna nr:5 och nr:6 valdes att frikopplas från dess motliggande komponent. Syftet med detta var att tillåta flexibilitet i spännkraft på remmen som kopplarihop de olika axlarna.

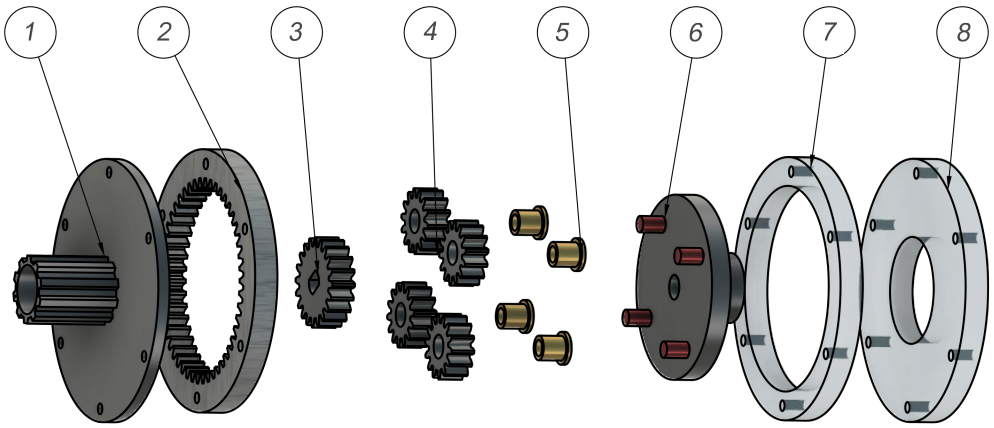
För att varje axel ska ha maximal kontaktyta med remmen var målet att skapa en likformig triangel mellan plan 1, plan 2 och drivaxeln. Det av just denna anledningen plan 1 lutar ut från plattformen medans plan 2 lutar in i mot plattformen.

Del nr:3 var från början endast tänkt att fungera som ett lagrat stöd till drivaxeln. Detta för att skydda den från spännkraften i remmen. Men då axelns utformning gjorde det omöjligt att föra på ett kullager valdes det att dela axeln i två. Detta tog bort möjligheten till fyrhjulsdrift.

Planetväxeln

Planetväxeln är uppbyggd av 8 olika komponenter. Kugghjulen är tillverkade i metall för att få en hållbar kraftöverföring. Växelhuset är tillverkat i akrylplast för att göra de inre komponenterna synliga.

Nr	Komponent
1	Bottenplatta
2	Ytterring
3	Solhjul
4	Planethjul
5	Bussningar
6	Planetbärare
7	Distans
8	Ändplatta



Figur 3.8: En översiktlig vy av planetväxelns olika komponenter.

Bottenplattan innesluter dels de invändiga kuggarna, samt fungerar den som en kraftöverföring till remmen. Komponenten är 3D-printad och har en kuggprofil med en tanddelning på $5,08\text{mm}/0,2$. Kuggprofilen är ihålig för att MG1s drivaxel ska nå solhjulet i planetväxeln. det sitter en lagring mellan bottenplattan och drivaxeln.

Bottenplattans skruvhål är försänkta för att undvika att skruvhuvudena kolliderar med remmen.

Samtliga kugghjul är tillverkade genom vattenskärning, det vill säga solhjulet, planethjulen och ringhjulet. Kugghjulen är raka spårväxlar med evolventprofil av modul 1, där solhjulet och planethjulet har utvändiga kuggar, medans ytterringen har invändiga kuggar. Solhjulet har ett hål med 2 flata kontaktytor för att få en fast ihopkoppling med dess drivaxel.

Planethjulet har dessutom ett brotschat hål på 6 mm. Vilket innebär att hålet får väldigt små toleranser, detta för att bussningarna ska passa glappfritt.

Bussningarna är tillverkade i mässing, de har en inre diameter brotschad till 4 mm. Bussningarnas syfte är att agera glidlager mellan planethjulen och planetbäraren.

Planetbäraren har fyra axlar av silverstål som bär planethjulen och glider mot mässingsbussningarna. Planetbäraren har även ett presspassat kullager som stöder ändplattan, och en stoppskruv för att låsas mot sin drivaxel.

Distansen och ändplattan är laserskurna ur akrylplast, distansen skapar utrymme för planetbäraren och ändplattan bär upp hela växelhuset då den är lagrad mot planetbäraren.

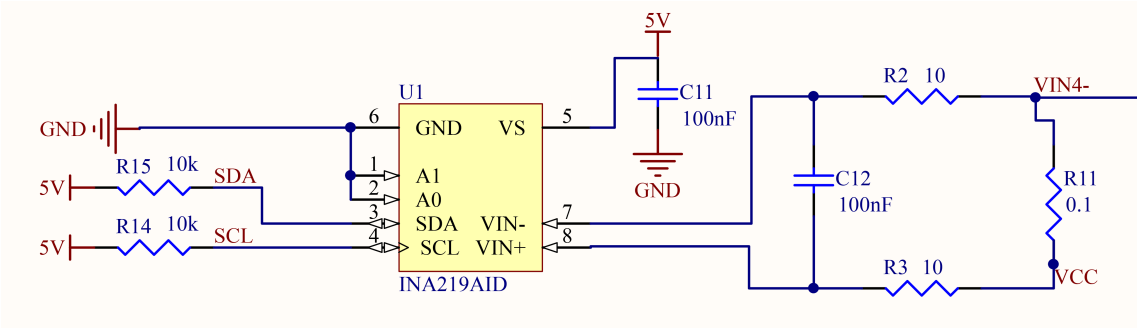
3.2 Elektronik

För att kunna göra ett reglersystem åt drivlinan måste motorernas varvtal samt strömstyrkan in i motorerna mätas med sensorer. Styrkorten Betaflight 32Bit 35A 2-6S BLHeli_32 ESC Dshot1200 valdes av just den anledningen att de har intern strömmätning och varvtalsmätning. Förhoppningen var att den informationen skulle vara tillgänglig genom UART för en Arduino som ska styra motorerna. Efter flertalet misslyckade försök att upprätta kommunikation mellan styrkorten och Arduinon valdes det i stället att mäta ström och varvtal med externa sensorer. Eftersom det finns tre motorer där båda mätningar måste göras individuellt beslutades det att designa ett kretskort för att samla och organisera alla komponenter.

Strömmätning

Två olika grundprinciper för att mäta ström övervägdes: induktiv och resistiv mätning. Då sensorerna kommer sitta i en miljö med roterande magnetfält från flera motorer kan de påverkas om de är av induktiv typ. Därför valdes en resistiv sensortyp. Eftersom tiden är begränsad valdes Texas Instruments INA219, då en gruppmedlem hade tidigare erfarenheter med det chippet.

I databladet för INA219 finns en referensdesign. Då inga särskilda krav fanns valdes denna design för att spara tid. Designen fungerar så att chippet INA219 mäter spänningsfallet som uppstår över shuntmotståndet R11 då en ström går genom motståndet. Eftersom spännigfallet är proportionellt mot strömmen genom motståndet så kan strömmen räknas ut, givet att motståndet är känt. Den här informationen filtreras internt, och skickas sedan ut på I2C-bussen. Kopplingsschemat från databladet [18] ritades upp i Altium Designer, och synes i figur 3.9. Detta implementerades sedan på ett kretskort.



Figur 3.9: INA219 referensdesign. Komponent R11 är shuntmotståndet, genom vilket strömmen leds igenom. Spänningsfallet över det motståndet (som är proportionellt mot strömmen) är vad INA219 mäter.

Varvtalsmätning

För att mäta motorernas varvtal togs dessa metoder i beaktning: rotationsenkoder på motoraxeln, optisk mätning (likt den lösning som finns i datormöss med kula), hallsensor nära motorns rotor och mätning av pulstakten på motorns faser.

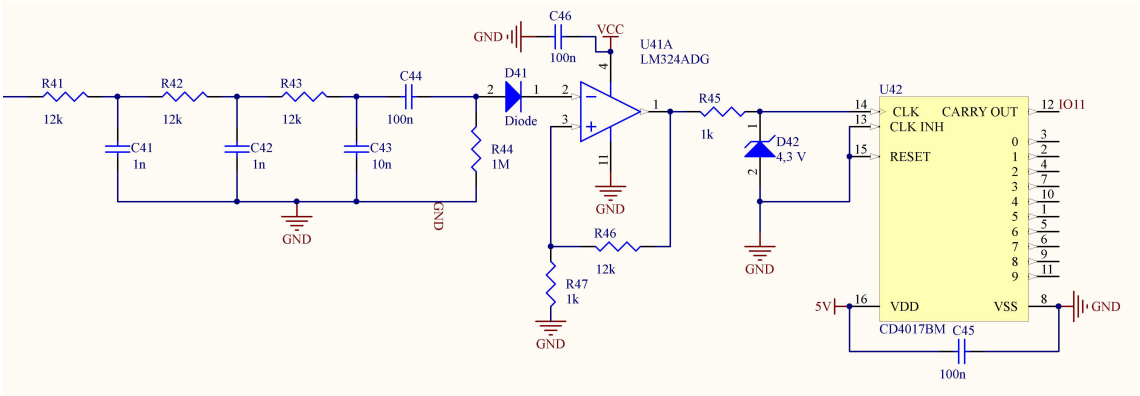
Ett av målen var att samla så mycket som möjligt av de nödvändiga komponenterna på ett och samma kretskort för att underlätta den mekaniska designen och konstruktionen. Detta gjorde att lösningarna med rotationsenkodrar gick bort till fördel för att mäta direkt på motorns fas då den lösningen endast kräver en sladd mellan motor och sensor. Eftersom motorerna är synkronmotorer så är varvtalet de roterar med proportionellt mot frekvensen på spänningen som motorerna matas med. Alltså finns informationen om motorns varvtal att att hämta från faserna.

Idén var att genom att lågpassfiltrera den hackiga fyrkantsvågen från styrkorten erhålla en sinusformad signal. Denna signal skulle sedan förstärkas, då amplituden är svag på låga varvtal, och sedan skickas in i en räknare som skickar ut en fyrkantsvåg med en tiondels frekvens av den inkommande signalen. Detta gör att det inte genereras för många avbrott i Arduinon.

Kretsen designades först på kopplingsplatta baserat på ovanstående idé. För att verifiera resultatet observerades signalen med oscilloskop. Det visade sig tidigt att ett första ordningens lågpassfilter inte var tillräckligt för att jämna ut den hackiga signalen från styrkorten. Ett tredje ordningens lågpassfilter konstruerades, och signalen blev sinusformad men med en DC-förskjutning. För att centrera signalen kring 0 V infördes även ett högpassfilter. För att ta bort den undre halvan av signalen sattes en likriktardiod på signalens utgång. Signalen förstärks därefter av en operationsförstärkare med 12 gångers förstärkning så att en svag signal från en motor på lågt varv fortfarande kan avläsas. Därefter används en Schottky-diod för att begränsa signalen till 4,7 V. Detta skickas sedan in i en räknare som skickar ut en puls för var tionde inkommande pulser. Pulstakten ut från räknaren räknas sedan av en Arduino, och omvandlas i mjukvara till varvtal genom att räkna ut skillnaden i tid mellan pulserna.

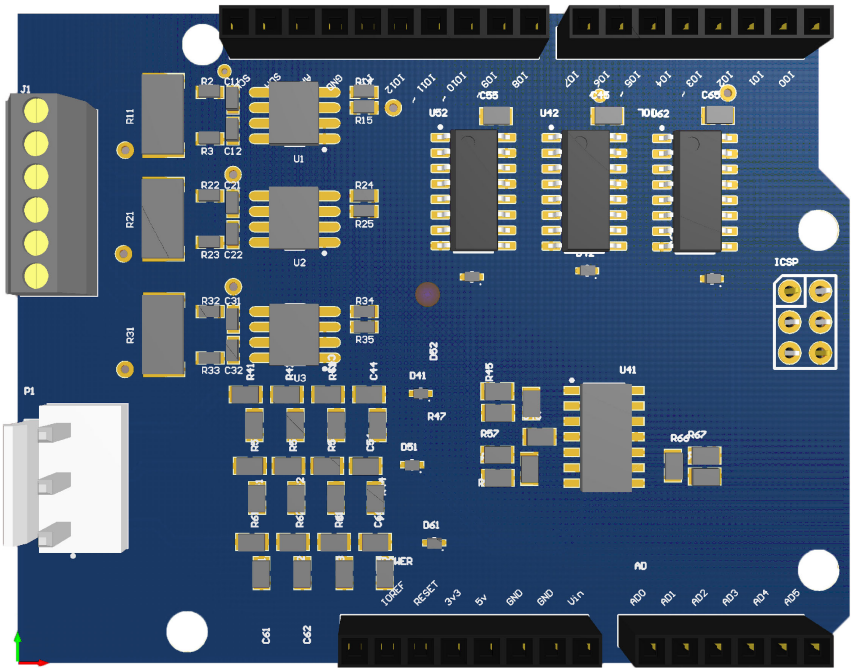
Implementation på kretskort

Kopplingschemat från den konstruerade kretsen på kopplingsplattan antecknades för att senare ritas upp i Altium Designer. Även referensdesignen för INA219 ritades upp i datorn. Eftersom det kommer finnas tre motorer på bilen måste det även finnas tre kanaler av varje sensortyp.



Figur 3.10: Schematisk bild av varvtalsmätande krets. Signalen från motorn kommer in som en hackig, brusig våg i R41 och bandpassfiltreras till något sinusaktigt, varpå den likriktas i D41. Därefter förstärks signalen och skickas in i U42 utifrån vilken var tionde puls tas.

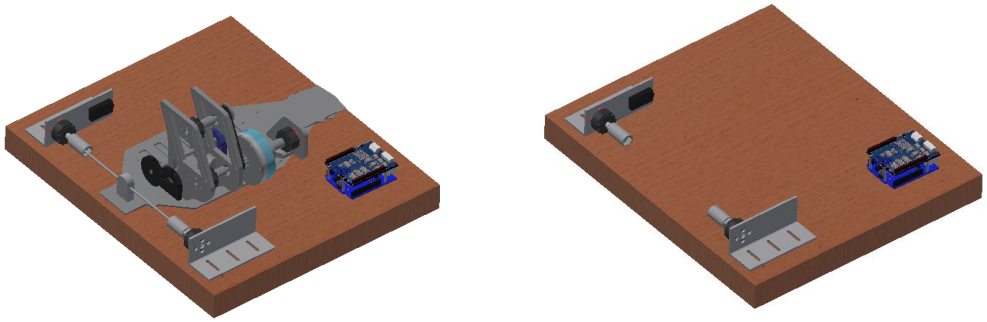
Med kopplingsschemat framställt kunde därefter en kretskortsdesign framställas. Eftersom detta kort är tänkt att staplas på en Arduino Uno användes en mall för att minimera mängden manuellt arbete [17]. Den färdiga designen skickades till PCBWay för tillverkning.



Figur 3.11: Det färdiga kretskortet sett ovanifrån. I den övre vänstra kvadranten sitter de tre strömmätande kretsarna. I den nedre vänstra kvadranten sitter bandpassfiltret. I nedre högra kvadranten sitter operationsförstärkaren. I övre högra hörnet sitter räknarkretsarna.

3.3 Testbänk

Syftet med testbänken är att kunna utföra tester på drivlinan under en kontrollerad miljö. Genom att generera konstlaster, både positiva och negativa, kan testbänken simulera olika körcykler. I figur 3.12 illustreras testbänken med drivlinan monterad.



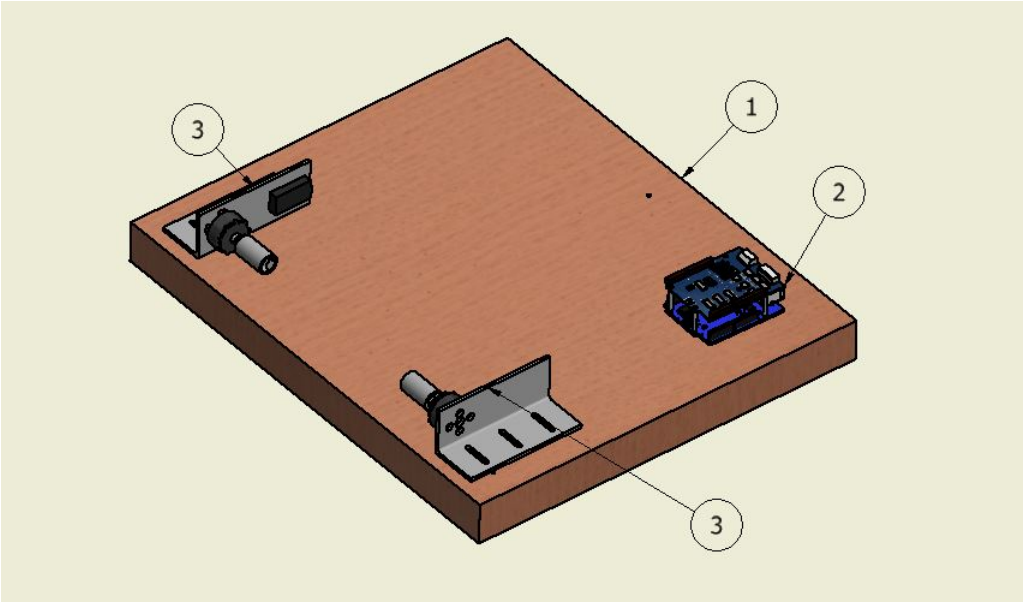
Figur 3.12: Till vänster i bilden illustreras testbänken med drivlinan monterad. Till höger är endast testbänken synlig

Testbänkens uppbyggnad

För att utföra tester på drivlinan behövdes det en testbänk för att fästa chassit. För att generera en konstlast krävs ställbara effektkomponenter med ett mätsystem för att samla data om komponenternas drift. Det behövs en fästanordning för att fästa effektkomponenterna till bottenplattan, såväl som en koppling mellan effektkomponenterna och drivlinan.

För konstlasten användes likadana elmotorer och styrkort som i drivlinan. Elmotorerna jobbar antingen med eller mot drivlinan för att generera bromsande eller accelererande moment. Drivlinans bakre axel kopplades till en elmotor på vardera sida chassit via två hjulnav. Mätsystem och styrkort användes på samma sätt som i drivlinan, med en Arduino och ett extra kretskort som mäter effekt och styr motorerna.

Nr	Komponent
1	Bottenplatta
2	Styr- och mätsystem
3	System för simulerad konstlast



Figur 3.13: En översiktlig bild av testbänken och dess komponenter.

Bottenplatta

Syftet med bottenplattan är att ge en bas till fordonet, motorerna, mätsystemen, styrsystem, och en yta där dessa kopplas ihop. Bänken kan göras på flera olika sätt. En träplatta har använts med följande motivering.

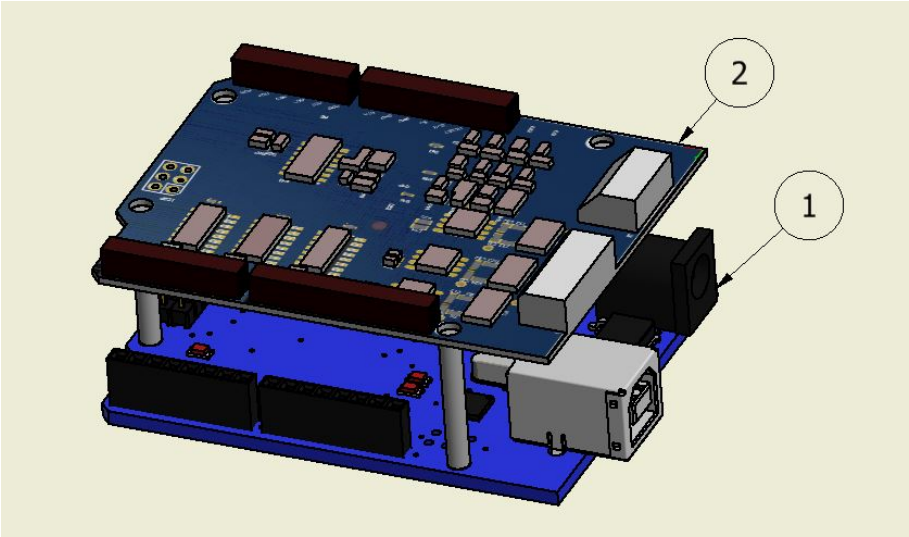
- Ekonomi: Testbänkens bidrag till projektet är begränsad, onödiga kostnader bör undvikas och alltså bör plattan vara billig.
- Material: För att vara anpassningsbart och lättbearbetat bör inte plattan vara för hård eller svårarbetad.
- Storlek: Plattan behöver vara stor nog för att rymma all komponenterna som ska fästas på plattan.
- Stabilitet: När systemets samtliga 5 elmotorer jobbar tillsammans kommer vibrationer uppstå, med en stabil bottenplatta kan dessa minimeras och systemet hålls på plats.

Styr- och mätsystem

För att styra motorerna används motorstyrkorten Betaflight 32Bit 35A2-6S BLHeli_32 ESC Dshot1200, alltså likadana som i drivlinan. De är kopplade till en Arduino med ett kretskort utformat för att mäta varvtal på samma sätt som beskrivs i avsnitt 3.2.

Styr- och mätsystems komponenter

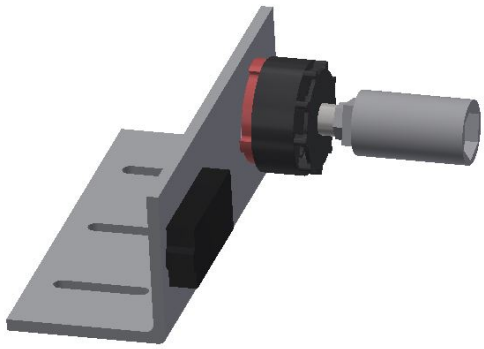
Nr	Komponent	Kommentar
1	Styrkort av testbänken	Arduino, som i drivlinan
2	Mätkort	Kretskort som i drivlinan



Figur 3.14: Styrkortet av testbänken som Arduinon styr motorerna och mätkortet mäter effekter.

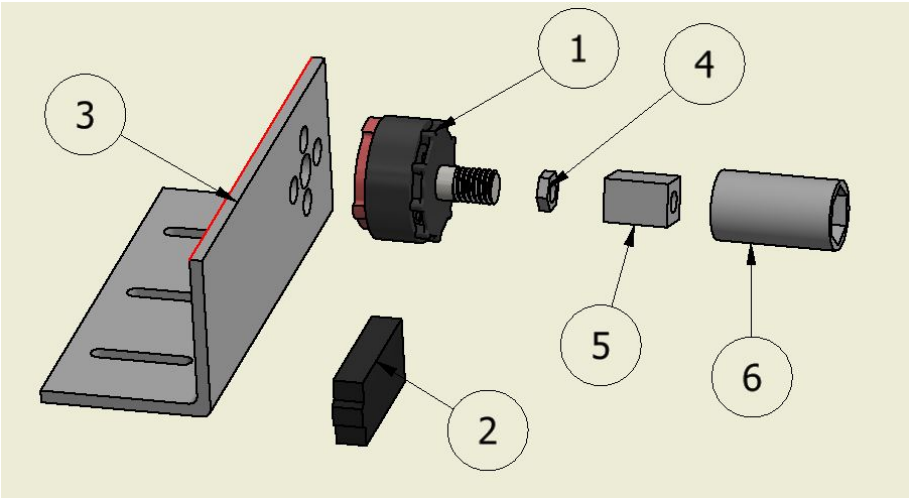
System för simulerad konstlast

För att simulera olika körcykler har följande system tillverkats. Det består av sex stycken komponenter som tillsammans kopplas på drivlinans bakre hjulnav för att utöva de motstånd som behövs för den simulerade körcykeln. Två systems används, ett på var sida om chassit.



Figur 3.15: Motorfäste med elmotor, styrkort, låsmutter, distans och sexkantshylsa. Hylsan kopplas på drivlinans bakre hjulaxel. Tillsammans bildar dessa komponenter ett system som kan användas för att simulera olika körcykler på drivlinan.

Nr	Komponent
1	Elmotor
2	Styrkort
3	Motorfäste
4	Låsmutter
5	Distans
6	Sexkanthylsa



Figur 3.16: En översiktlig bild på samtliga komponenter som bildar systemet för simulerad konstlast.

1. Elmotor: Drivlinans två bakre hjulnav kopplas till testbänken, alltså behövs det två stycken elmotorer. Motorernas effekt kan varieras både med- och moturs för att simulera olika körscenarion.

2. Styrkort: Båda elmotorerna behöver varsitt styrkort för att kontrollera motorn. Elmotorn kopplas direkt till styrkortsutgången. Likadant styrkort användas

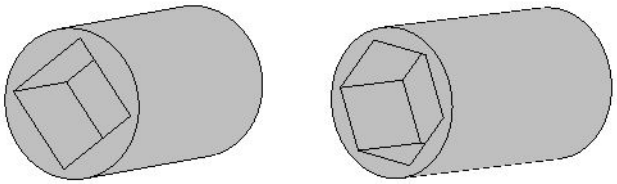
som i drivlinan.

3. Motorfäste: Syftet med delen är att fästa elmotorn och styrkortet på bottenplattan med en viss flexibilitet vid installationen. En sida av motorfästet fästs på träplattan med skruvar. För att lätt kunna montera motorfästena på ett flexibelt sätt så har 3 spår frästs ut på sidan som skruvas mot testbänken. Detta gör att man lätt kan göra små justeringar, och ta loss chassit genom att lossa på skruvarna och skjuta isär motorfästena. På den andra sidan sitter elmotorn och styrkortet. Drivlinans bakaxel och elmotorn måste ligga i linje, vilket leder till att toleranserna för elmotorns fästpunkter är små.

4. Låsmutter: Då distansen är fäst på motoraxeln med gängor så kommer den skruva ut sig när motorn roterar antingen medurs eller moturs. Alltså behövs något för att låsa fast distansen på motoraxeln. I drivlinan användes stoppskruvar för att uppnå detta resultatet då utrymmet var begränsat. Då det finns mer utrymme på testbänken så valdes det att använda en låsmutter. Låsmuttern skapar en konstant spänning genom att den skruvas åt motsatt håll gentemot distansen, så att de jobbar mot varandra. Då motoraxeln är 13 mm med endast 10 mm som är gängat, så valdes det att fräsa låsmuttern till 2 mm vilket är halva dess ursprungliga storlek.

5. Distans: Komponenten agerar koppling mellan motoraxeln och hylsan. Eftersom elmotorerna har en axel med utvändiga gängor, så går det inte att koppla motorerna direkt till drivlinans hjulnav. Distansen har ett centrerat hål med M5 gängor för att passa över elmotorns hjulaxel. Distansen är 15 mm lång och har en kvadratisk sida med sidlängden 10 mm. Detta för att kunna passa i ett 10 mm fyrkantsfäste.

6. Sexkantshylsa: Syftet med hylsan är att koppla ihop testbänken med drivlinan. Hylsan har ett sexkantsmönster som passar på drivlinans hjulnav, och ett 10 mm fyrkanstfäste på andra sidan som passar på distansen.

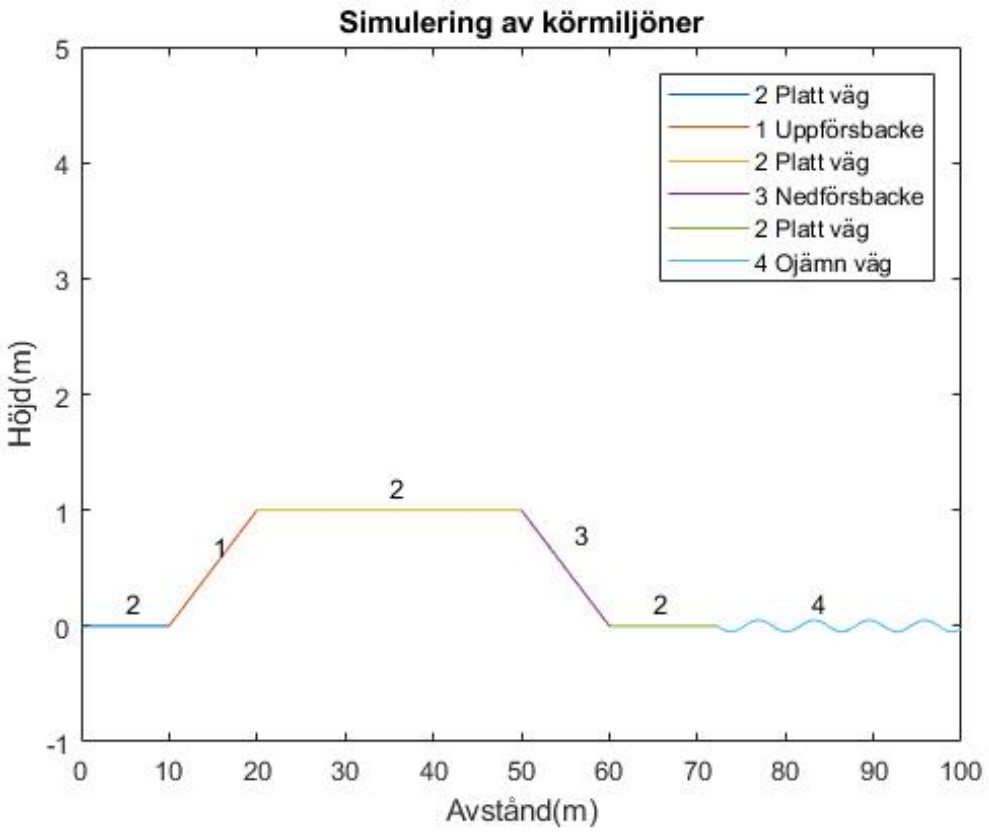


Figur 3.17: Sexkantshylsan som kopplar ihop testbänken med drivlinan. Dess fyrkantsfäste som fästs på distansen syns i den övre bilden, medans den hexagonala sidan kopplas på drivlinans hjulnav vilket illustreras i den undre bilden.

Simuleringsmiljö

Testbänkens elmotorer används för att simulera olika körcykler. Genom att driva motorerna antingen med eller mot drivlinan kan både uppförs- och nedförsbackar simuleras. Genom att simulera en nedförsbacke medans drivlinans motorer inte driver kan man simulera en motorbromsning, och på så sett testa den regenerativa funktionen på styrkorten. I figur 3.18 visas ett exempel på en körcykel, med fyra olika belastningar.

1. Uppförsbacke: Friktion av vägen plus fordonets tyngdkraftskomponent
2. Platt väg: Motor genererar vägfriktion
3. Nedförsbacke: Friktion av vägen minus fordonets tygkraftskomponent
4. Ojämn väg: En blandning av de tre ovan.



Figur 3.18: En illustration av de olika körförhållandena testbänken behöver kunna simulera för att simulera en körcykel.

3.4 Grafiskt gränssnitt

För att kommunicera med bilen och Arduinon framställdes det ett grafiskt gränssnitt syftat till att hjälpa användaren att avläsa information från Arduinon samt att ställa in inställningar och styra bilen elektroniskt direkt från gränssnittet. Informationen som lästes in från Arduinon presenterades i from av grafer för att göra informationen lättläslig. Tre grafer i gränsnittsfönstret visade information om effekt, varvtal och gasreglage. Språket som användes för programmet var Java då det besitter goda verktyg för framställning av grafiska gränssnitt som var relativt lätta att använda samtidigt som det fanns tidigare erfarenhet inom gruppen av detta språk.

Koncept

Tidigt i arbetet begränsades de mest essentiella egenskaperna av gränssnittet till följande punkter:

- Grafer som visar information från från bilen som ska levereras med hjälp av en Arduino.
- Ett sätt att justera inställningar till hårdvaran elektroniskt.
- Att kunna gasa med olika kraft med hjälp en funktion i det grafiska gränssnittet.
- Eventuellt ge möjlighet till olika förinställingar.

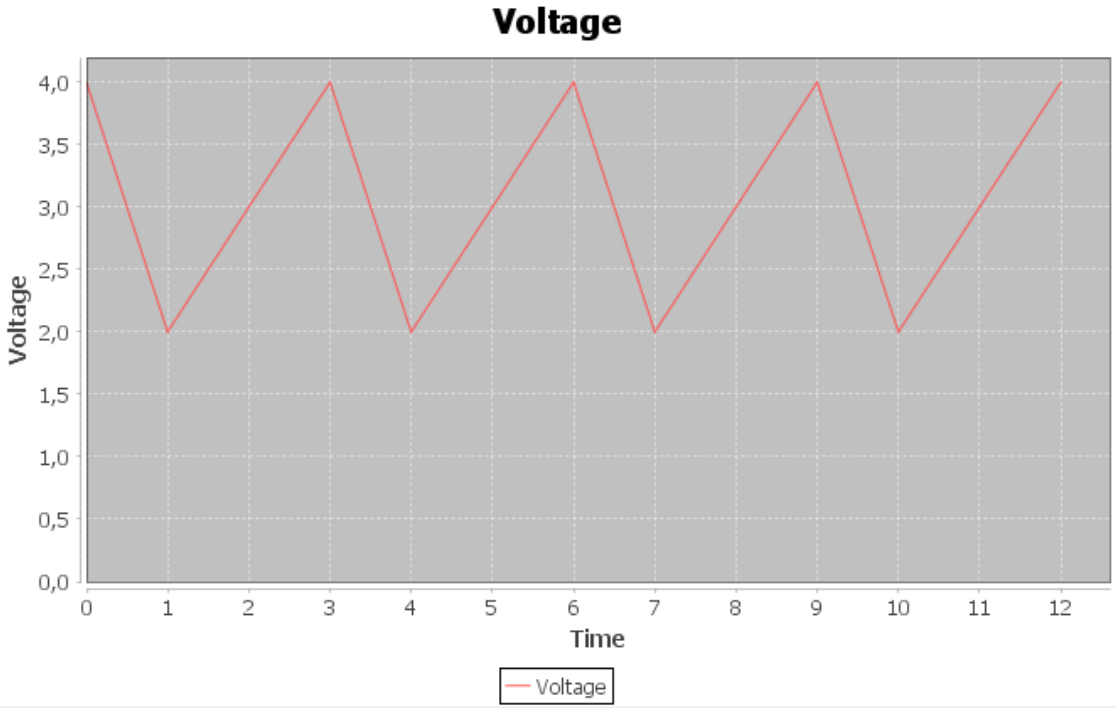
De flesta av dessa komponenter kunde framställas relativt smidigt med hjälp av Javas inbyggda standardbibliotek. De mer utmanande bitarna var implementeringen av Arduino-kommunikation och graferna eftersom att Java saknar stöd för dessa två komponenter i sitt standardbibliotek. Detta behövde lösas genom att importera externa bibliotek för Java som hade stöd för dessa funktioner. För att kontinuerligt uppdatera graferna krävdes det att gränssnittet kunde läsa information från Arduinon och för att kontrollera motorerna var det nödvändigt att koden kunna skicka data till Arduinon.

Bortsett från möjligheten av kontroll av bilen skulle graferna ta högsta prioritet och därför skulle majoriteten av utrymmet på gränssnittet också allokeras till dem så att de förblev tydliga och lättlästa för användaren som utförde testerna.

Implementering av grafer

Javas standardbibliotek saknar stöd för den typen av grafer som avsågs användas i projektet vilket ledde till att ett externt bibliotek var tvunget att användas [14]. JFreeChart valdes för att det enkelt tillät implementationen av de sökta graferna samt att det stödjer statistikföring [15].

Grafen läser in ett dataset, vilket är en grupp av olika flyttalsströmmar, och ritar sedan upp dessa för att illustrera deras relation samt göra det enklare för användaren att avläsa och tolka mätningarna. Graferna hade inga problem med höga uppdateringsfrekvenser och visade förändring väldigt tydligt med dess röda linje.



Figur 3.19: En JFreeChart-graf som visar en kontinuerlig indata-strömning av olika flyttal.

I bilden 3.19 är X-axeln feldefinierad i och med att den är märkt som *Time*, men i verkligheten mäter den inte tid utan uppdateringscykler som har passerat sedan start. Graferna sköter expandering av axlarna automatiskt om det matas in ett nummer som är större än vad axlarna tillåter för tillfället. Om det skulle behövas har användaren även möjligheten att zooma in eller ut på specifka delar av grafen för att kunna se specifika delar mer i detalj.

Rent tekniskt använder graferna sig av ett antal dataset där det lagras flyttal från Arduinon och som sedan ritas upp i grafen. Den följer alltså en ganska vanlig metod för hur grafer brukar ritas upp, men skillnaden är att graferna i gränssnittet gör allt i realtid samtidigt som bilen kör och uppdaterar sina värden.

Implementering av Arduino-kommunikation

För att kunna kombinera hårdvaran och mjukvaran alls behövdes det möjliggöras för ihopkopplingen av båda parter så att dessa skulle kunna kommunicera med varandra. Funktionen att läsa in från Arduinon användes till att skicka information till graferna så att resultaten skulle kunna illustreras tydligt. Eftersom att en av gränssnittets deluppgifter var att kunna agera som en digital ”gaspedal” till bilen var det nödvändigt att skrivning till Arduinon från gränssnittet var möjligt för att digitalt kunna kontrollera varje separat motor på bilen.

För Arduino-kommunikation i Java-koden krävs det stöd från ytterligare ett externt bibliotek vilket innehåller funktioner som tillåter gränssnittet att koppla upp sig mot en Arduino samt skriva till och läsa ifrån den. Det externa biblioteket *JSerialComm* ger användargränssnittet förmågan att koppla upp sig mot Arduinon för att tillåta kommunikation och sedan med hjälp av de inbyggda funktionerna är det möjligt att extrahera den information som söks.[16] Den inbyggda funktionen *SerialPort.getCommPorts()* söker efter anslutbar hårdvara i systemet och när hårdvaran har funnits går det att ansluta till den med en enkel knapp.

Läsa från Arduinon

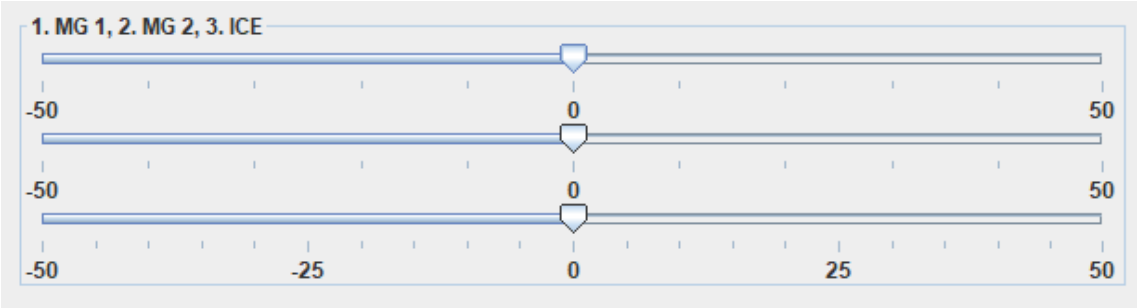
När Arduinon kopplas upp mot datorn och den får ström börjar den genast skicka ut information som den läser av hårdvaran till dess indata-ström. Gränssnittets jobb är att fånga upp texten som Arduinon skriver ut till strömmen, kasta bort de oviktiga bitarna och sedan skriva ut dem till graferna. Eftersom att denna delen behövde vara kompatibel med graferna innebar det många extra steg för att allt skulle kunna funka tillsammans.

För att kunna läsa in något från indata-strömmen alls implementerades först en *Scanner* i gränssnittets kod som tillät inläsning av indatan. *Scannern* fungerar så att den kallar på en mening i taget från indatan och använder en specifik algoritm för att veta vilka delar i indatan som är relevanta för graferna och vilka delar som kan kastas bort. Algoritmen använder sig av kunskap över hur strukturen på indatan ser ut för att kunna filtrera bort de oviktiga delarna och endast spara det som mär intressant.

Eftersom att det som Arduinon skriver ut som indata är av typen *String* måste gränssnittet hantera det genom att konvertera de relevanta värdena till flyttal för att de ska vara kompatibla med graferna som ska illustrera dem. Konverteringen sker enkelt och snabbt med hjälp av funktionen *Float.parseFloat();* efter att alla värden har separerats från resten av indatans text och skickas sedan in till grafens dataset.

Skriva till Arduinon

Processen att skriva information till Arduinon från gränssnittet utfördes på liknande sätt, fast med rollerna omkastade. I detta fall var det istället gränssnittets uppgift att skicka information till utdata-strömmen som Arduinon sedan kunde läsa in och sedan allokera flyttalen till rätt motorer och få dem att snurra i begärd hastighet. Det behövdes implementeras ett digitalt reglage för att tillåta användaren att kunna vara mer exakt i vilka flyttal som skickades in i Arduinon. Detta reglage blev alltså den ”digitala gaspedalen” i slutändan.

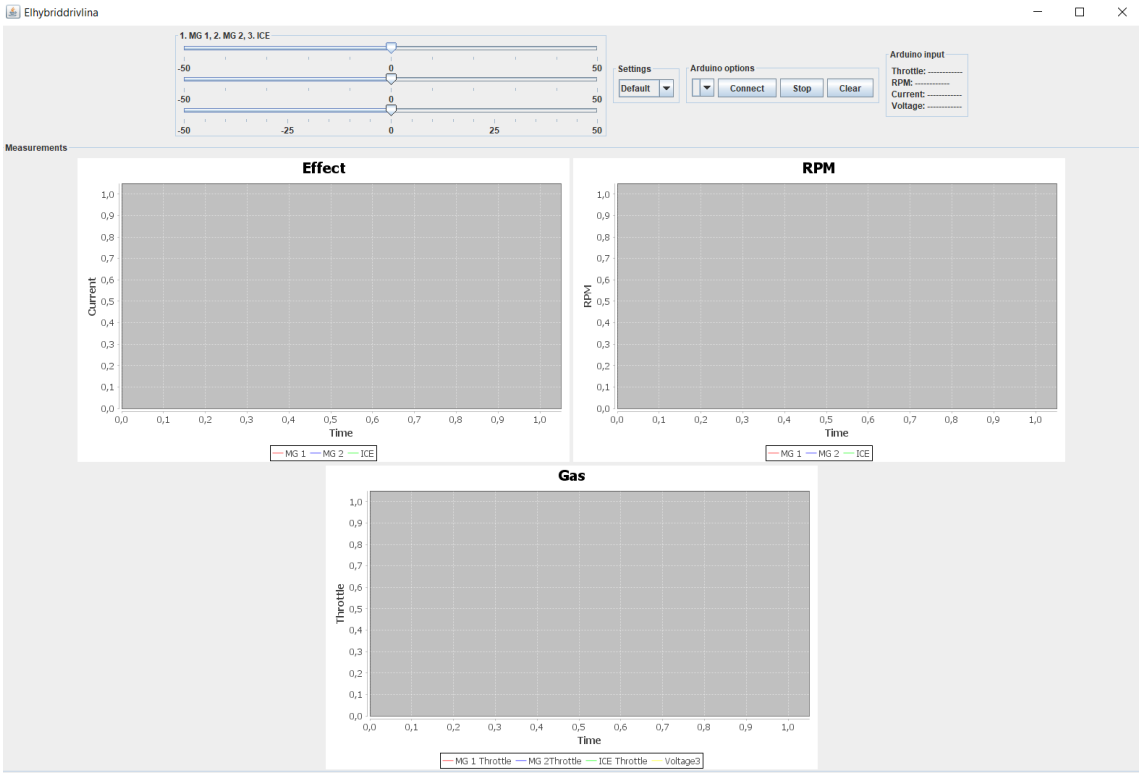


Figur 3.20: De implementerade reglagen som ska agera som ”digitala gaspedaler” för bilens tre motorer; ICE, MG1 och MG2. Dessa är uppbyggda av JSliders och genom en ekvation kan dessa mata ut flyttal som Arduinon sedan kan läsa in.

På samma sätt som Arduinon skriver till gränssnittet med typen *String* skriver också gränssnittet till Arduinon med samma typ. Detta innebar att på samma sätt som gränssnittet konverterar till flyttal behövde Arduinon kunna göra samma sak. När användaren justerar de digitala reglagen så skickas det en lång *String* till Arduinon som innehåller alla nya värden för motorerna som användaren önskar förändra. Arduinon läser sedan in denna och börjar separera alla värden från varandra så att varje hamnar i en individuell variabel istället för en lång *String*. När separationsprocessen är klar kan Arduinon sedan konvertera alla värden till flyttal genom funktionen *.toFloat();* för att sedan skicka de nya flyttalen till respektive motor. Här gäller samma princip både för att kontrollera bilens motorer, men också för att kontrollera testbänkens motorer.

Den färdigställda mjukvaran

Efter många olika prototyper av gränssnittet som antingen fallerade på designval eller att funktioner inte fungerade lyckades det i slutändan framställas en version som kunde framställa de resultat som söktes och var tillräckligt användarvänligt.

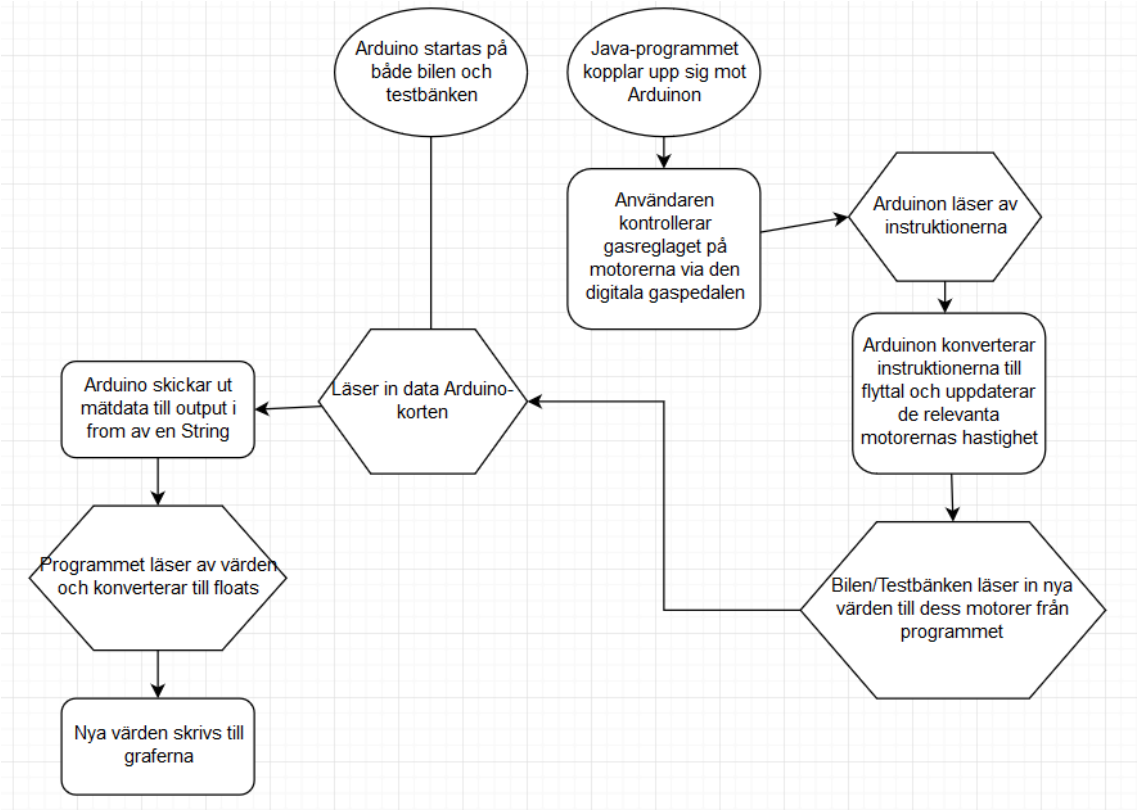


Figur 3.21: Den slutgiltiga versionen av det grafiska gränssnittet. Många av de gamla komponenterna som var överflödiga har tagits bort eller ändrats. Informationsrutan är förändrad och flyttat till det övre högra hörnet och har gjorts smidigare.

I det slutgiltiga gränssnittet lades det till tre grafer som vardera visar olika mätningar, vilket är effekt, varvtal samt gasreglage. Alla tre tar in mätvärden från varje motor så att varje graf illustrerar tre olika linjer, en för varje motor. Utöver det så implementerades också möjligheten för anslutandet till Arduinon för att etablera kommunikation. Programmet hittar vilket port Arduinon är kopplad till automatiskt och användaren behöver endast trycka på *Connect*-knappen för att ansluta och börja läsa av eller att skriva. Eftersom att korten fungerar på liknande sätt för både bilen och testbänken innebär det också att uppkopplingen för båda till gränssnittet fungerar på liknande sätt.

I det övre vänstra hörnet lades det till reglage som är till för att kontrollera hur mycket kraft som bilens motorer ska köra med och åt vilket håll vardera motor snurrar mot. Med hjälp av dem kan användaren köra bilen och kontrollera vad som ska testas. Textrutan har också uppdaterats till att vara smidig och lättläst.

Både graferna och kontrollering av bilen sker genom sändning av flyttal både till och från Arduinon. Eftersom att inläsningen av indata-strömmen och skrivningen till indata-strömmen underlättas betydligt av att informationen sker i typen *String* skickas alla flyttal som ska uppdateras på en gång i en stor *String* och konverteras sedan i mjukvaran till flyttal. När Arduinon läser av nya värden från bilen skickas informationen i form av en sekvens av *Strings* som skickas till gränssnittet där de delas upp och konverteras till flyttal och sedan ritas upp av graferna för att ge användaren en tydlig bild av prestanda.



Figur 3.22: Flödesschema på hur gränssnittet fungerar i samband med Arduino-kortet samt hur de arbetar parallellt med varandra.

4

Resultat

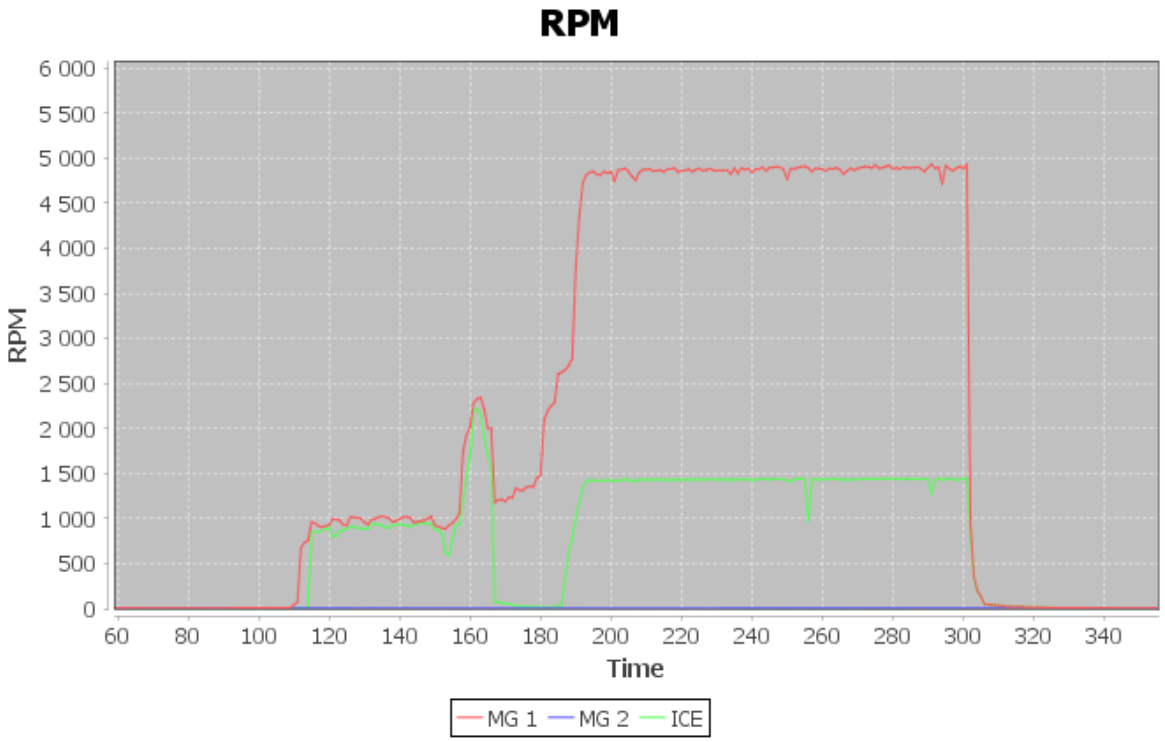
I detta kapitlet redovisas resultaten från projektet. När drivlinan körs i testbänken kan mjukvaran generera grafer som visar de olika motorernas hastighet över tid. I del 4.1 verifieras mätsystemens funktionalitet genom att jämföra dessa grafer med de ekvationer som beskriver drivlinan i teorikapitlet. Del 4.2 visar ett körscenario där det syns hur motorerna påverkar varandra. Resultatet blev mycket begränsat på grund av ett fel i motorerna, mer om det tas upp i Diskussion.

4.1 Verifiering

För att verifiera drivlinan samt mätutrustningen gjordes ett test. Genom att låta MG1 rotera i godtycklig hastighet samtidigt som ringhjulet är låst, så ska ICE rotera i en viss hastighet enligt ekvation 2.3. Resultatet från testet visas i figur 4.1. MG1 håller en stabil hastighet på strax under 5000 rpm under en lång period, under denna perioden är ringhjulet låst. Under samma period roterar ICE med strax under 1500 rpm. Genom att använda MG1:s genomsnittliga rpm tillsammans med dimensioneringen av planetväxeln enligt figur 3.3 i ekvation 2.3, så fås den teoretiska hastigheten för ICE enligt följande

$$\omega_c = \frac{4900 * 20 + 0 * 48}{48 + 20} = 1441. \tag{4.1}$$

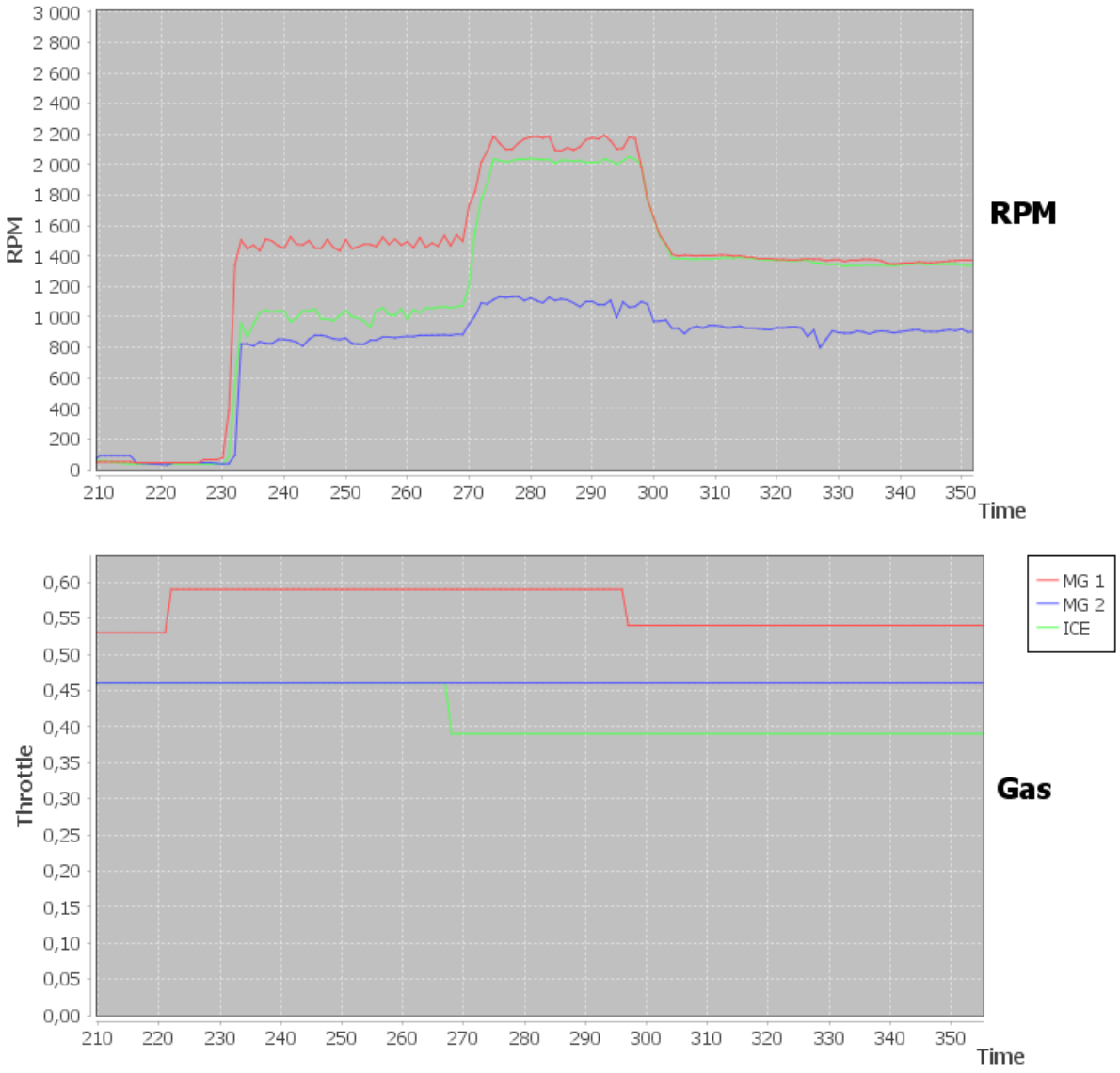
Det teoretiska värdet stämmer väl överens med vad som syns i graf 4.1. Detta visar att planetväxelns dimensionering ger den utväxling som var förväntad, det visar dessutom att mätsystemen ger pålitliga värden.



Figur 4.1: Varvtalen för MG1 och ICE under ett test. Under perioden då de båda motorerna har en relativt stadig hastighet är ringhjulet låst, vilket innebär att motorernas varvtal är direkt beroende av varandra.

4.2 Körscenario

Ett fall som undersöktes var *start från stillastående*. Scenariot motsvarar en bil som accelererar ifrån ett stillastående läge. Detta görs genom att till en början driva med MG1 för att nå en hastighet där ICE kan verka. När ICE väl driver så slutar MG1. I figur 4.2 redovisas resultaten som uppmättes när drivlinan körde scenariot *start från stillastående* i testbänken. I grafen *Gas* så står 0.50 för ingen gas, medans 0.00 är maximal gas åt ena hållet och 1.00 är maximal gas åt det andra. För att mätsystemet ska registrera varvtalen så krävs det att det går lite ström genom motorerna. Därför sätts gasen till 0.47 samt 0.53. Detta är den minimala mängden gas som behövs för att mätsystemet ska registrera varvtalen, men ger inget betydande vridmoment till motorerna.



Figur 4.2: Varvtalen och gasen från scenariot *start från stillastående*. Då MG2 är direkt kopplat till chassits drivaxel kan dess graf ses som en icke skalenlig representation av hastigheten.

Eftersom att MG2 är direkt kopplad till chassits drivaxel, och därmed direkt till hjulen, så kan det ses som en icke skalenlig representation av hastigheten. Till en början syns det hur MG1 sätter drivlinan i rörelse, och drar med sig ICE till cirka 1000 rpm. I denna hastigheten kan ICE börja driva. När ICE sedan börjar driva syns det att ICE drar med både MG1 och MG2 till högre varvtal. Vad som är intressant är det som händer när MG1 slutar driva, det syns tydligt hur ICE:s varvtal sjunker med MG1:s.

5

Diskussion och Slutsats

I detta kapitel kommer de olika delarna av projektet att diskuteras, fokus kommer ligga på problem som gruppen stött på under arbetet samt förbättringar som kan göras för att bättre uppnå målen som satts.

5.1 Anledning till svagt resultat

Motorerna Emax RS2205S 2300KV ”Red Bottom” RaceSpec Motor som användes för drivlinan valdes av den anledningen att det har lägst varvtal av alla motorer i prisklassen 100 kr. 2300KV betyder att de snurrar i varvtalet 2300 rpm/V utan belastning. Många av de andra motorerna i prisintervallet 50 till 250 kr har ett KV värde mellan 3000 och 5000, vilket innebär att de snurrar fortare, men med lägre vridmoment. Tväremot förväntningarna visade experimenten att motorerna inte verkade leverera något som helst vridmoment vid låga hastigheter, något som var mycket överraskande då elmotorer i regel levererar stort vridmoment i låga hastigheter. Den enda motorn som kan driva bilen på egen hand är MG1 men den behöver fortfarande hjälp av mänsklig knuff i startögonblicket. ICE och MG2 kan endast driva när de kör tillsammans med MG1. När planetväxeln var helt nyligen inoljad var friktionen överhuvudtaget inte nämnvärd vid drift av solhjulet. I detta läge kunde MG1 på egen hand få drivaxeln att rotera och resultaten som visats tidigare i rapporten erhölls. Detta begränsade kraftigt möjligheten att utföra simuleringar i projektets slutfas varpå resultaten och möjligheten att uppnå målen således blivit lidande.

En förklaring till detta som tagits upp inom gruppen är att motorerna som köptes in är designade för drönare och helikoptrar. Det enda motståndet en helikoptermotor måste övervinna är luftmotståndet, om då motståndet plötsligt blir mycket högt måste det betyda att helikoptern har krashat eller någons hand som har åkt in i en propeller. Således är hypotesen att en inbyggd säkerhetsfunktion hos motorstyrkorten begränsar strömmen till motorerna när den känner för stort motstånd. Funktionen finns dokumenterad i online-manualen för mjukvaran för styrkorten som ”Low RPM Power Protection” [20]. Detta stärker teorin samt att gruppen obser-

verat en oförändrad strömtillförsel till motorerna trots ökad effekt vid stillastående körning.

5.2 Drivlina

Chassit inhandlades tidigt i projektet, då den avgörande faktorn var priset, när det i själva verket borde varit storleken. Alltså inhandlades ett billigt men relativt litet ($190 * 365$ mm) chassi. Vilket ledde till att storleken på varje komponent har varit ett överhängande bekymmer som försvårat designprocessen. Detta har medfört att drivlinan har blivit väldigt kompakt, vilket försvårar montering och eventuella finjusteringar. Ett problem som enkelt hade kunnat undvikas genom att köpa ett större chassi.

Planetväxeln var från början till större delen tillverkad i laserskuren akrylplast. Den enda komponenten som var tillverkad i metall var planetbäraren. Friktion var närvarande men inte speciellt stor i denna planetväxel. När bilens alla komponenter för första gången monterades och första testkörningen startades kunde inte motorerna få hjulen att rotera. Då det vid detta stadiet inte fanns några misstankar om ett fel i motorerna beslöts det att istället tillverka planetväxeln i metall för att minska friktionen. Detta var en tidskrävande process som med facit i hand kanske inte hade behövt göras. Men det ska trots allt nämnas att friktionen var lägre i den nya planetväxeln. Till eventuell fortsatt utveckling av denna drivlina rekommenderas det att undersöka möjligheten att ta bort den trodda säkerhetsfunktionen som nämndes i 5.1.

Till en början var planen att tillverka drivlinans samtliga hållare i metall. Detta arbetet påbörjades men visade sig vara mycket tidskrävande och vara väldigt känsligt för små misstag under tillverkningsprocessen. Att gå över till 3D-printade komponenter har minskat tillverkningshastigheten drastiskt. Och möjliggjort en iterativ prototypande arbetsgång, då en ny prototyp med små justeringar kunnat printas över natten.

5.3 Testbänk

Testbänken blev mekaniskt färdig men då mjukvaran för att styra dess motorer aldrig färdigställdes saknas den funktionalitet som sattes upp som mål i början av projektet. Med ett tillhörande mjukvarusystem är testbänken ett effektivt verktyg för att testa liknande drivlinors effektivitet.

5.4 Mjukvara

Mjukvaran i helhet gör det den ska och samarbetar bra med hårdvaran för att göra det lätt för användaren att kontrollera bilen och läsa av dess prestanda. I slutändan lyckades inte alla de planerade delarna implementeras i slutversionen, särskilt grafen som var menad att visa på relationen mellan varvtal och effekt. Utöver det lyckades de mest viktiga delarna implementeras in i gränssnittet och den lyckades producera grafer från datan som lästes in från Arduinon.

En av de mer uppenbara problemen med kommunikationen mellan gränssnittet och Arduinon var att stundvis kunde uppkopplingen av kommunikation mellan de två parterna sluta upp osynkroniserade. Detta var ett resultat av att Arduinon börjar samla och skicka information till utdata-strömmen så fort den får ström. Detta betydde att gränssnittet kunde kopplas upp till Arduinon vid en icke önskvärd tidpunkt i utströmningen av data och det kunde leda till att datavärdena hamnade fel i graferna. Oftast var detta inte ett större problem i och med att det gick och hitta rätt efter en omstart av programmet.

Ytterligare ett problem som uppstod i processen av utvecklingen av mjukvaran var att den ett antal gånger behövde undergå stora designförändringar på grund av bristande erfarenhet av gränssnittsdesign inom delgruppen som hade ansvar för det. Det ledde till att utvecklingen ofta stod still då mycket tid behövde läggas på att kasta om komponenter och skriva om kod på nytt för att allt skulle se rätt ut. Mycket av den bristande erfarenheten låg i importeringen och användingen av externa bibliotek som var nödvändiga för att skapa och implementera graferna, men också för att sköta kommunikationen med Arduino-kortet. I slutändan fungerade allting väldigt bra ihop, med processen dit inkluderade en hel del svårigheter och mycket extra arbete.

Trots att det var tillräckligt med arbetskraft att ha en ensam person på utvecklingen av mjukvara hade det varit en positiv förändring att utse en person att agera som bollplank för idéer om design och peka ut funktioner som glömts eller som skulle kunna läggas till för ett bättre program. Om feedback kommer kontinuerligt hela tiden under projektets gång sparas det mer tid när alla delgruppen samlas ihop för att kombinera alla delar och minskar risken att det saknas en viktig funktion.

5.5 Mätningar

Ett av målen med projektet var att kunna mäta effektåtgång och varvtal hos alla motorer genom de kretsar och den mjukvara som designats. Resultatet av dessa mätningar är de grafer som presenterats tidigare, men körcykler kunde inte implementeras då detta hade krävt ytterligare mjukvara vilket inte var möjligt inom

tidsramen. De mätningar som kan utföras visar dock på systemens stabilitet och metodernas korrekthet. På grund av anledningarna som nämnts i 5.1 spenderades mycket tid på att försöka få drivlinan att fungera vilket inte gav möjlighet att skriva mjukvara till testbänken. Graferna i resultatkapitlet visar dock hur grunden har lagts för ett efterkommande projekt att bygga på.

5.6 Miljö- och samhällsaspekter

I och med att hybrida bilar släpper ut mindre avgaser än rena bensibilar kan det anses att det genomförda projektet ligger i linje med utvecklingen av ett mer hållbart samhälle då hybriddrivlinor kan gå delvis på el istället för enbart på bensin. Eftersom att fossila drivmedel är en ohållbar lösning ligger det i samhällets intresse att försöka hitta en alternativ lösning till de områden som fortfarande använder sig av dessa fossila bränslen[21]. Ett alternativ som har börjat visa sig på marknaden är företag som producerar bilar som drivs helt på el som bränsle, t.ex Tesla. Dock är i dagsläget priset på dessa alternativ fortfarande för höga eller saknar vissa fördelarna som bensinbilarna besitter som körsträcka och produktion för att riktigt kunna slå sig in på marknaden än. Infrastrukturen som krävs för att försörja en full övergång till elbilar är inte heller riktigt på plats än.

Av denna anledning kan hydriddriften agera som en brygga över till ett samhälle där bilar inte längre drivs av fossila bränslen. Kombinationen av el- och bensinmotor i hybriddirvlinan mjuknar det annars hastiska utbytet av bilbränsle i samhället och skapar mer tid för samhället att utveckla det el-nät som kommer krävas för elbilarna samtidigt som avgaserna minskas. Hybriddrivlinan har många av elbilens fördelar men kan samtidigt täcka dess svagheter genom att använda sig av bensinmotorn när det behövs. Det fungerar som en kompromiss för konsumenter som vill vara miljövänliga men ändå behöver kunna köra långsträckor frekvent.

I och med att elmotorn väsnas väldigt lite kommer det med en stor nackdel, vilket är att det är svårare för människor att höra en bil som närmar sig. I de flesta situationer kan brummandet av en vanlig bensinbil vara en god varningssignal för personer som ska gå över vägen för att veta att en bil är på väg och att det då kan vara nödvändigt att titta omkring, speciellt för personer med synskador. Detta är en varningssignal som försvinner med elmotorer i och med att de är så tysta i jämförelse med bensinmotorer. Av denna anledning har det nu börjat forskas om möjligheten för bilar som använder sig av elmotorer att spela upp ett artificiellt ljud vid körning för att uppmärksamma personer i närheten [22].

5.7 Slutsats

Målen som sattes i början av projekt har visats sig i efterhand vara för ambitiösa. Att skapa grunden som i sin tur möjliggör utförandet till många av de uppsatta målen var mer tidskrävande är vad som från början förväntades. Trots detta ser inte gruppen arbetet som misslyckat. Den grunden som skapades visar på goda möjligheter för vidareutveckling för eventuella kommande kanidatprojekt. Det hade varit intressant att undersöka möjligheten att applicera den redovisade metoden för minimering av bränsleförbrukning på drivlinan på vår uppställda modell. Dock hade ett reglertekniskt system behövt ställas upp för att modellera Priusens motsvarigheter enligt de varvtals- och vridmomentsbegränsningar som råder.

Litteraturförteckning

[1] Larminie J. & Lowry J. (2003) Electric Vehicle Technology Explained, Kapitel 1: Introduction (s.1-21) ISBN 0-470-85163-5

[2] Dr. Eberhardt James J. (2002) Energy Efficiency and Renewable Energy https://www1.eere.energy.gov/vehiclesandfuels/pdfs/deer_2002/session1/2002_deer_eberhardt.pdf

[3] Wiseman, E. (2018) What's the difference between a hybrid, a plug-in hybrid, and an electric 'EV' car? https://www.telegraph.co.uk/cars/advice/difference-hybrid-plug-in-hybrid-electric-ev-car/

[4] Alternative Fuels Data Center U.S. Department of Energy (hämtad mars 2018) https://www.afdc.energy.gov/vehicles/electric_basics_hev.html

[5] Alternative Fuels Data Center U.S. Department of Energy (hämtad mars 2018) https://www.afdc.energy.gov/vehicles/electric_basics_phev.html

[6] Volvo Laddhybrider https://www.volvocars.com/se/bilar/upptack/laddhybrider (hämtad mars 2018)

[7] Prius Drivlina THS II http://www.toyota-global.com/innovation/environmental_technology/technology_file/hybrid/hybridsystem.html (hämtad mars 2018)

[8] Liu, W.(2017) Hybrid Electric Vehicle System Modeling And Control s. 73-75

[9] Concepcion, M. (2011) Advanced Hybrid Vehicle Systems s.40-43

[10] Jinming Liu, Huei Peng (2008) Configuration, Sizing and Control of Power-Split Hybrid Vehicles https://pdfs.semanticscholar.org/fd95/5287801b56338185326cb26d08b282316eb5.pdf

[11] Murgovski et al. (2017?) Computationally efficient energy management of a planetary gear hybrid electric vehicle

[12] Naidu, D. (2003) Optimal Control Systems s.69

[13] Oracle *Oracle Java Documentation*, 2017.
https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

[14] Oracle *Oracle Java Documentation*, 2017
https://docs.oracle.com/javase/tutorial/deployment/selfContainedApps/addlibrary.html

[15] JFreeChart *Welcome to JFreeChart*, 2017
http://www.jfree.org/jfreechart/

[16] JSerialComm *What is jSerialComm?*, 2018
https://fazecast.github.io/jSerialComm/

[17] samarkh *Altium UNO shield template*, 2012
https://forum.arduino.cc/index.php?topic=115446.0

[18] Texas Instruments *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface*, 2012
http://www.ti.com/lit/ds/symlink/ina219.pdf

[19] Modell av Toyotas planetväxel *Toyota Prius - Power Split Device (PSD)*, 2015?
http://eahart.com/prius/psd/

[20] Operation Manual For BLHeli_32 http://www.blheli32.com/operation-manual-for-blheli_32/ (hämtad mars 2017)

[21] Environmental and Energy Study Institute *Fossil Fuel*, 2018
http://www.eesi.org/topics/fossil-fuels/description

[22] NCBI *Impact of adding artificially generated alert sound to hybrid electric vehicles on their detectability by pedestrians who are blind*, 2012
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3396425/

A

Källkod Java

Direkta källkoden kan hittas på: `https://github.com/Pompero/Chai/tree/master/GraphTest`

```
import java.awt.BorderLayout;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.XYPlot;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

import com.fazecast.jSerialComm.SerialPort;

import java.awt.event.*;
import java.io.PrintWriter;
import java.util.Scanner;
import java.awt.*;

public class GraphTest {

static SerialPort chosenPort;

//Ints for Sliders and such
static int throttle1int = 0;
static int throttle2int = 0;
static int throttle3int = 0;
static int RPM1int = 0;
static int RPM2int = 0;
static int RPM3int = 0;
```

```
static int current1int = 0;
static int current2int = 0;
static int current3int = 0;
static int voltageint = 0;

//Ints for the sliders
static int alittle = -50;
static int amiddle = 0;
static int alot = 50;
static int fifty = 50;

//Floats for printing.
static float motor1int = (float)0.5;
static float motor2int = (float)0.5;
static float bensinint = (float)0.5;
static float randomfloat = 1;

public static void main(String[] args) {

// Creating and Configuring window.
JFrame window = new JFrame();
window.setTitle("Elhybriddrivlina");
window.setSize(1600, 1080);
window.setLayout(new BorderLayout());
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Creating components
JComboBox<String> portList = new JComboBox<String>();
JButton connectButton = new JButton("Connect");
JButton stopButton = new JButton("Stop");
JButton clearButton = new JButton("Clear");
JPanel centerPanel = new JPanel();

//Top panel
JPanel topPanel = new JPanel();
JPanel topPanelw = new JPanel();
JPanel bigtopPanel = new JPanel();
JPanel topPanele = new JPanel();

//Sliders for motorkontroller
JSlider bensin = new JSlider(JSlider.HORIZONTAL, alittle, alot, amiddle);
JSlider motor1 = new JSlider(JSlider.HORIZONTAL, alittle, alot, amiddle);
JSlider motor2 = new JSlider(JSlider.HORIZONTAL, alittle, alot, amiddle);

//motor1
motor1.setMajorTickSpacing(50);
```

```
motor1.setMinorTickSpacing(10);
motor1.setPaintTicks(true);
motor1.setPaintLabels(true);

//motor2
motor2.setMajorTickSpacing(50);
motor2.setMinorTickSpacing(10);
motor2.setPaintTicks(true);
motor2.setPaintLabels(true);

//bensin
bensin.setMajorTickSpacing(25);
bensin.setMinorTickSpacing(5);
bensin.setPaintTicks(true);
bensin.setPaintLabels(true);

//Get Value from sliders (temporary)
String motor1value = Integer.toString(motor1.getValue());
String motor2value = Integer.toString(motor2.getValue());
String bensinvalue = Integer.toString(bensin.getValue());

motor1.addChangeListener(new ChangeListener() {

@Override
public void stateChanged(ChangeEvent arg0) {
JSlider motor1 = (JSlider)arg0.getSource();
if(!motor1.getValueIsAdjusting()) {
motor1int = (float)0.5 - (((float)motor1.getValue())/100);
}
}
});

motor2.addChangeListener(new ChangeListener() {
@Override
public void stateChanged(ChangeEvent arg1) {
JSlider motor2 = (JSlider)arg1.getSource();
if(!motor2.getValueIsAdjusting()) {
motor2int = (float)0.5 - (((float)motor2.getValue())/100);
}
}
});

bensin.addChangeListener(new ChangeListener() {
@Override
public void stateChanged(ChangeEvent arg2) {
JSlider bensin = (JSlider)arg2.getSource();
```

```
if(!bensin.getValueIsAdjusting()) {
bensinint = (float)0.5 - (((float)bensin.getValue())/100);
}
}
});

topPanele.setLayout(new BoxLayout(topPanele, BoxLayout.Y_AXIS));
topPanele.add(motor1);
topPanele.add(motor2);
topPanele.add(bensin);
topPanele.setBorder(BorderFactory.createTitledBorder("1. MG 1, 2. MG 2, 3
topPanele.setPreferredSize(new Dimension(600, 150));
bigtopPanel.add(topPanele);

//Text panel in the center (debug)
JLabel console = new JLabel();

//Side panels
JPanel westPanel = new JPanel();
JPanel eastPanel = new JPanel();
JPanel textPaneltop = new JPanel();

// Adding to the top panel.
topPanel.add(portList);
topPanel.add(connectButton);
topPanel.add(stopButton);
topPanel.add(clearButton);
//topPanel.add(console);
console.setText("<html>Throttle: ---------- <br>RPM: ---------- <br>C
topPanel.setVisible(true);
topPanel.setBorder(BorderFactory.createTitledBorder("Arduino options"));
textPaneltop.add(console);
textPaneltop.setBorder(BorderFactory.createTitledBorder("Arduino input"))

// TopPanel west
String[] presets = {"Default", "Preset 2", "Preset 3"};
JComboBox presetBox = new JComboBox(presets);
topPanelw.add(presetBox);
topPanelw.setBorder(BorderFactory.createTitledBorder("Settings"));

// Adding to big top panel
bigtopPanel.add(topPanelw);
bigtopPanel.add(topPanel);
bigtopPanel.add(textPaneltop);

// Add the top JPanel with the buttons and the drop-down box.
```

```
window.add(bigtopPanel, BorderLayout.NORTH);
window.add(centerPanel, BorderLayout.CENTER);

// Populate Drop-down box
SerialPort[] portNames = SerialPort.getCommPorts();
for(int i = 0; i < portNames.length; i++)
portList.addItem(portNames[i].getSystemPortName());

// Create line graph - Current
XYSeries current1 = new XYSeries("MG 1");
XYSeries current2 = new XYSeries("MG 2");
XYSeries current3 = new XYSeries("ICE");
XYSeriesCollection dataset = new XYSeriesCollection();
dataset.addSeries(current1);
dataset.addSeries(current2);
dataset.addSeries(current3);
JFreeChart chart1 = ChartFactory.createXYLineChart("Effect", "Time", "Cu
XYPlot currentPlot = (XYPlot) chart1.getPlot();

// Create line graph - RPM
XYSeries RPM1 = new XYSeries("MG 2");
XYSeries RPM2 = new XYSeries("ICE");
XYSeries RPM3 = new XYSeries("MG 1");
XYSeriesCollection dataset2 = new XYSeriesCollection();
dataset2.addSeries(RPM3);
dataset2.addSeries(RPM1);
dataset2.addSeries(RPM2);
JFreeChart chart2 = ChartFactory.createXYLineChart("RPM", "Time", "RPM",

// Create line graph 3 - Voltage and throttle
XYSeries voltage1 = new XYSeries("Voltage3");
XYSeries throttleseries = new XYSeries("MG 1 Throttle");
XYSeries throttle2series = new XYSeries("ICE Throttle");
XYSeries throttle3series = new XYSeries("MG 2Throttle");
XYSeriesCollection dataset3 = new XYSeriesCollection();
dataset3.addSeries(throttleseries);
dataset3.addSeries(throttle3series);
dataset3.addSeries(throttle2series);
dataset3.addSeries(voltage1);
JFreeChart chart3 = ChartFactory.createXYLineChart("Gas", "Time", "Throt

// Add the chart in the middle.
centerPanel.add(new ChartPanel(chart1));
centerPanel.add(new ChartPanel(chart2));
centerPanel.add(new ChartPanel(chart3));
centerPanel.setBorder(BorderFactory.createTitledBorder("Measurements"));
```

```
//Configure the connect button and use another thread to listen to data
connectButton.addActionListener(new ActionListener() {

@Override
public void actionPerformed(ActionEvent arg0) {
if(connectButton.getText().equals("Connect")) {
chosenPort = SerialPort.getCommPort(portList.getSelectedItem().toString()
chosenPort.setComPortTimeouts(SerialPort.TIMEOUT_SCANNER, 0, 0);
chosenPort.setBaudRate(115200);
if(chosenPort.openPort()) {
connectButton.setText("Disconnect");
portList.setEnabled(false);
}

//new thread that listen for incoming text and populates the graph
Thread thread = new Thread() {
@Override public void run() {
//Scanner
Scanner scanner = new Scanner(chosenPort.getInputStream());
while(scanner.hasNextLine()) {
try {
        scanner.useDelimiter("\\s[a-z]\\D");
//System.out.println(scanner.nextLine());
//Reading throttle
System.out.println(scanner.nextLine());
String throttle = scanner.nextLine();
//System.out.println(throttle);
float throttlenumber = Float.parseFloat(throttle);
throttleseries.add(throttle1int++, throttlenumber);
String throttle2 = scanner.nextLine();
//System.out.println(throttle);
float throttle2number = Float.parseFloat(throttle2);
throttle2series.add(throttle2int++, throttle2number);
String throttle3 = scanner.nextLine();
//System.out.println(throttle);
float throttle3number = Float.parseFloat(throttle3);
throttle3series.add(throttle3int++, throttle3number);

//Reading RPM
System.out.println(scanner.nextLine());
String RPM1line = scanner.nextLine();
float RPM1number = Float.parseFloat(RPM1line);
//System.out.println(RPM1line);
RPM1.add(RPM1int++, RPM1number);
String RPM2line = scanner.nextLine();
```

```
//System.out.println(RPM2line);
float RPM2number = Float.parseFloat(RPM2line);
RPM2.add(RPM2int++, RPM2number);
String RPM3line = scanner.nextLine();
float RPM3number = Float.parseFloat(RPM3line);
//System.out.println(RPM3line);
RPM3.add(RPM3int++, RPM3number);

//Reading Current
System.out.println(scanner.nextLine());
String current1line = scanner.nextLine();
float current1number = Float.parseFloat(current1line);
current1.add(current1int++, current1number);
String current2line = scanner.nextLine();
float current2number = Float.parseFloat(current2line);
current2.add(current2int++, current2number);
String current3line = scanner.nextLine();
float current3number = Float.parseFloat(current3line);
current3.add(current3int++, current3number);

//Reading voltage
System.out.println(scanner.nextLine());
String voltageline = scanner.nextLine();
float voltagenumber = Float.parseFloat(voltageline);
console.setText("<html>Throttle:␣" + throttle + "␣" + throttle2 + "␣" + t
"<br>RPM:␣" + RPM1line + "␣" + RPM2line + "␣" + RPM3line +
"<br>Current:␣" + current1line + "␣" + current2line + "␣" + current3line
"<br>Voltage:␣" + voltageline +"</html>");
window.repaint();
} catch(Exception e) {}
}
scanner.close();
// Scanner close //
try {Thread.sleep(2000);} catch(Exception e){}

PrintWriter output = new PrintWriter(chosenPort.getOutputStream());

while(true) {
output.print(motor1int + "," + bensinint + "," + motor2int);
output.flush();
try {Thread.sleep(2000);} catch(Exception e){}
}
}
};
thread.start();
}
```

```
else {
chosenPort.closePort();
portList.setEnabled(true);
connectButton.setText("Connect");
}
}
});

stopButton.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
}
});

clearButton.addActionListener(new ActionListener(){
@Override
public void actionPerformed(ActionEvent arg0) {
throttleseries.clear();
throttle2series.clear();
throttle3series.clear();
RPM1.clear();
RPM2.clear();
RPM3.clear();
current1.clear();
current2.clear();
current3.clear();
voltage1.clear();
throttle1int = 0;
throttle2int = 0;
throttle3int = 0;
RPM1int = 0;
RPM2int = 0;
RPM3int = 0;
current1int = 0;
current2int = 0;
current3int = 0;
voltageint = 0;
}
});
//Apparently this needs to be far down.
window.setVisible(true);
        }
}
```