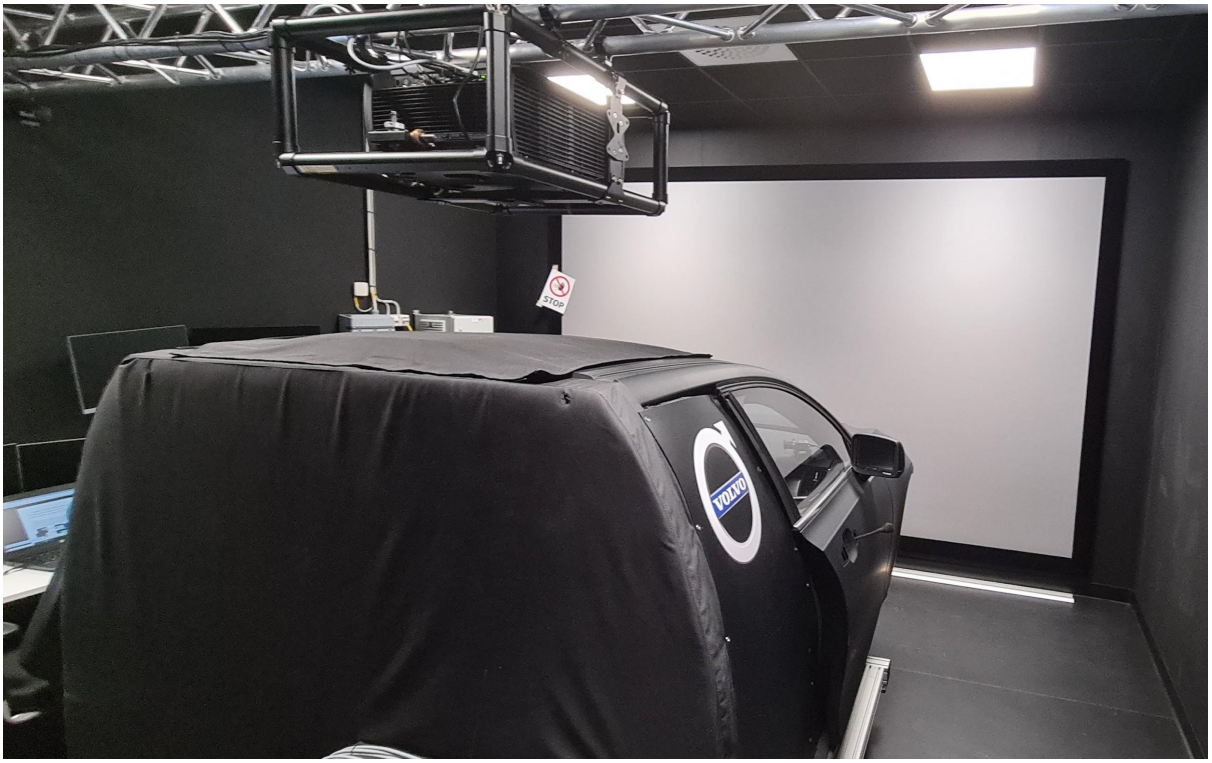
CHALMERS



Simuleringsverktyg med återkopplad styrning

Integrering av PC-baserad fordonssimulering och styrning med återkopplingshårdvara

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

JONAS SUNDBERG
ANTON LUNDIN

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2022
www.chalmers.se

Förord

Denna rapport är ett examensarbete genomfört på Volvo Cars, skriven av Anton Lundin och Jonas Sundberg, vid tiden studerande på Mekatronikprogrammet på Chalmers Tekniska Högskola. Arbetet har varit en lärorik resa där lösningar och förhoppningar inte alltid blivit som utgångsplaneringen. Projektet har krävt nytänkande för att uppnå alternativa lösningar och lösningsgångar. I dessa hinder har Matthijs Klomp på Volvo Cars givit stor stöttning och hjälp för att hålla arbetet igång och i rätt riktning. Vidare har han deltagit i utformning av de tester som genomförts. Nya lärdomar har fått inhämtas för nyttjande av programvaror som CarMaker och CANoe, där David Andersson från IPG Automotive och Fadi Khalaili från Vector givit svar på alla frågor som kom upp under arbetets gång. Sist vill vi även säga tack till Veronica Olesen på Chalmers som alltid varit snabb på att svara på frågor och funderingar.

Sammanfattning

Bilindustrin världen över använder mer och mer simuleringar för att utveckla nya produkter. Idag använder Volvo Cars sig av en realtidsdator för att köra ett fordonssimuleringsprogram. Genom att sammankoppla datorn med en fysisk testrigg, styrs bilratten i simuleringsprogrammet med den fysiska ratten i testriggen. Data skickas mellan testriggen och simuleringsprogrammet som bildar ett återkopplat system. Den här rapporten visar hur en PC med Windows kan användas i stället, samt hur det påverkar prestandan av simuleringen.

Arbetet är avgränsat till att endast testa EtherCAT och CAN som kommunikationsbuss. Vad som är mest ekonomisk utvärderas inte. Simuleringsprogrammets efterliknande av en riktig bil har inte utvärderats.

Både EtherCAT och CAN har undersökts och testats, där EtherCAT har körts med en tredjehands programvara medan en egen applikation har skapats för CAN. För dessa två alternativ har simuleringsprogrammet stabiliserats med ekvidistanta tidssteg i programkoden, detta för att kunna köras närmre hård-realtid. Resultatet visar att en realtidsdator är det bästa alternativet vid simulering, men att CAN via en PC kan vara ett alternativ.

Även teoretiska värden för styrenhetens PD regulator har tagits fram via tester och beräkningar. Dessa värden är grundade på att den fysiska ratten inte ska påverka styrkänslan, utan istället framhäva den simulerade ratten.

Abstract

The worldwide automotive industry is using more and more simulations to develop new products. Today, Volvo Cars uses a real-time computer for a physical test rig to control a steering wheel in a simulation. By connecting the computer and the physical test rig, the simulated steering wheel is controlled with the physical steering wheel. Data is sent between the test rig and the simulation program and therefore forms a feedback system. This report refers to how a PC with Windows can be used instead, as well as how it affects the performance of the simulation.

The work is limited to only test EtherCAT and CAN as communication buses. The most economical will not be evaluated, nor has the simulation program been evaluated regarding if it is life-like.

Both EtherCAT and CAN have been investigated and tested, where EtherCAT has been run with a third-party software while an integrated application has been created for CAN. For the simulation program to be able to run closer to hard real-time, it has been stabilized with equidistant timesteps in the program code. The results show that a real-time computer is the best option for simulation, but that CAN via a PC can be useful.

Theoretical values for the control unit's PD controller have also been calculated with test results. These values are based on the fact that the physical steering wheel should not affect the sense of steering and be small enough to represent the simulated steering wheel.

Innehållsförteckning

Terminologi/Förkortningar

Börvärde - Insignal till en regulator/system
CAN - Controller Area Network, är en databuss som oftast och främst används i fordon.
dbf-fil - Databas fil som innehåller definieringar av signaler
ESI - Extended Script Infrastructure. En konfigurationsfil.
HIL - Hardware in the loop (refererar främst till NI HIL som används i projektet)
Jitter - Tidsdifferenser mot ett önskat värde.
Little Endian - Bytevis skifta ett hexadecimalt tal
Master-to-slave - En huvudenhet (Master) som styr en eller flera slavenheter(Slave)
PCIe - Peripheral Component Interconnect Express
PDO - Process Data Object
QPC - QueryPerformanceCounter
RTD - Realtidsdator
SDO - Service Data Object
Testrigg - refererar till servomotor och styrenhet
Ärvärde - Utsignal från en regulator/system

1. Inledning

Denna rapport går igenom hur ett simuleringsprogram byggt för en realtidsdator (RTD) kan anpassas till en PC med Windows. Arbetet är gjort på Volvo Cars. En lösning av problemet bidrar till en mer användarvänlig version av deras existerande simulator, samt en förhoppning om att test- och simuleringsprogram ska kunna användas flitigare.

1.1 Bakgrund

Medan bilar blir mer tillgängligt för hela världen, ställs det högre krav för varje ny bilmodell. Det medför att mer tester behövs för att säkerhetsställa att allt fungerar som det ska innan lansering av nya bilmodeller. För att inte behöva testa lika mycket i praktiken med riktiga bilar, testas mycket med simuleringar. Detta medför mindre resursslöseri och möjligheten till att snabbt ändra inställningar som annars kräver ombyggnation. Tester blir även enklare repeterbara och generellt säkrare vid farliga manövrar.

Idag använder Volvo Cars hård realtid med hjälp av en RTD för att köra simuleringar med ett återkopplat system på en fysisk testrigg. Testriggen är till för att genomföra tester med en människa i simuleringsmiljön. RTD fungerar, men är inte optimalt eftersom det kräver extra kompetens och är en större kostnad jämfört med att använda en Windows PC. Volvo Cars använder även andra mjukvaror som är anpassade för PC, som inte kan användas på realtidsdatorn. Därför kommer arbetets fokus ligga på att hitta en lösning som kombinerar ett PC-baserat simuleringsverktyg med Volvo Cars nuvarande testrigg.

1.2 Syfte

Syftet är att testa och visa hur ett PC-baserat simuleringsprogram kan användas och sättas upp som styrningsåterkoppling till en testrigg. Vidare ska prestanda och användbarhet jämföras med en RTD.

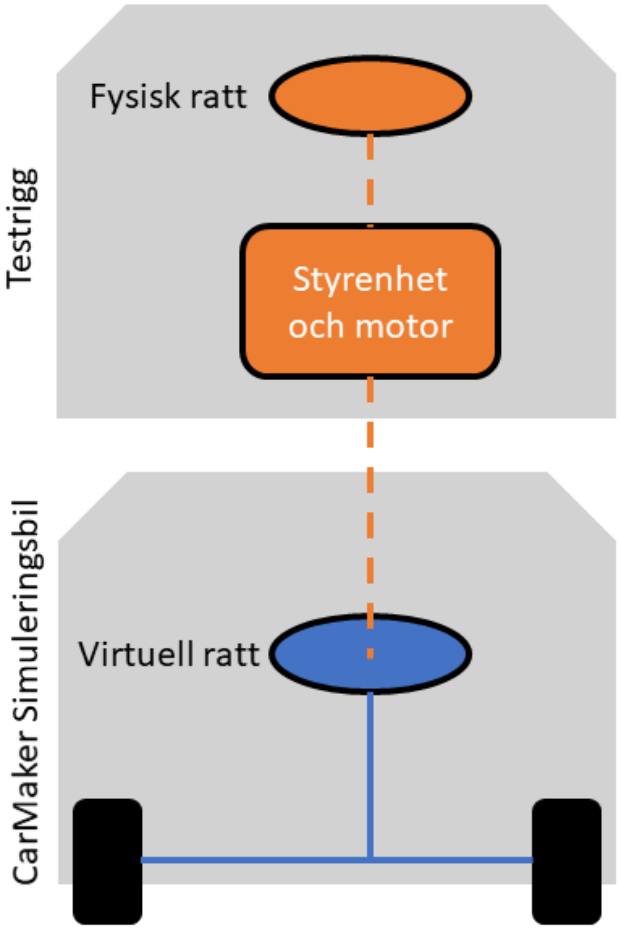
1.3 Avgränsningar

- Vad som är mest ekonomiskt utvärderas inte. Endast olika system testas med hänsyn till prestanda.
- Hur verklighetstrogen simuleringen är kommer inte utvärderas, utan simuleringens styrkänsla är slutmålet.
- EtherCAT och CAN kommer användas som kommunikation. Andra typer av kommunikationsprotokoll utvärderas ej.
- Arbetet avgränsas till användandet av en PC med specifikationer enligt bilaga 2.

1.4 Precisering av frågeställningen

Målet för arbetet är att få styrningsåterkoppling från en testrigg att fungera med det PC-baserade simuleringsprogrammet CarMaker. Jitter ska begränsas till låga nivåer för att få god mänsklig interaktion. Uppkopplingen ska ske via EtherCAT och CAN. Genom tester ska data tas fram och jämföras med hänsyn till prestanda. Målet är att den fysiska

ratten ska kännas som den simulerade. Figur 1 visualiserar en förlängd styrstång kopplad på den virtuella ratten, där rattstången är en testrigg med styrenhet och servomotor.



Figur 1: Visualisering av systemet mellan simulering och testrigg

Delmål:

- Sätt upp master-to-slave kontakt mellan PC:ns Ethernet till styrenheten i testriggen.
- Via ett EtherCAT-kort koppla samman PC:n med styrenheten i testriggen.
- Via ett Vector CAN Interface koppla samman PC:n med styrenheten i testriggen.
- Få den fysiska ratten att vara så lik som möjligt den simulerade ratten.

Frågeställning:

- Hur kan jitter minimeras i simuleringsprogrammet CarMaker?
- Hur är prestandaskillnaden mellan en RTD och en PC vid simulering?
- Hur är prestandaskillnaden mellan CAN och EtherCAT vid simulering?
- Hur påverkas styrkänslan av testriggens regulator?

2. Teoretisk / Teknisk bakgrund
Kapitlet omfattar teori, programvaror och hårdvaror för projektet.

2.1 Realtidssystem
Ett realtidssystem är ett system som reagerar inom ett förväntat tidsintervall efter att en händelse inträffat[1]. Det betyder att alla funktioner inom ett system hela tiden ska vara beredda på att exekveras. Att rätt funktioner exekveras under dess förväntad tid säkerställs genom två metoder, att sätta en prioriteringsordning, samt genom att sätta en schemaläggning. Prioriteringsordning innebär att olika funktioner inom systemet är olika viktiga. En funktions prioritering kan ändras beroende på hur länge den väntat på att bli exekverad. Schemaläggning inom systemet betyder att om en funktion ber om att bli exekverad, men kräver att andra funktioner måste exekveras först, finns det ett schema för sekvensen. Det säkerställer att en funktion alltid har rätt data att arbeta med.

Ett realtidssystem kan även innehålla fler än en enhet. Därför måste enheterna i systemet vara beredda på att reagera på varandra. Exempelvis om en enhet A reglerar temperaturen i ett rum och enhet B läser av temperaturen. Enhet A skickar en signal till enhet B för att läsa av den nuvarande temperaturen, som sedan skickas tillbaka till enhet A. Därefter har enhet A all den data som krävs för att reglera temperaturen.

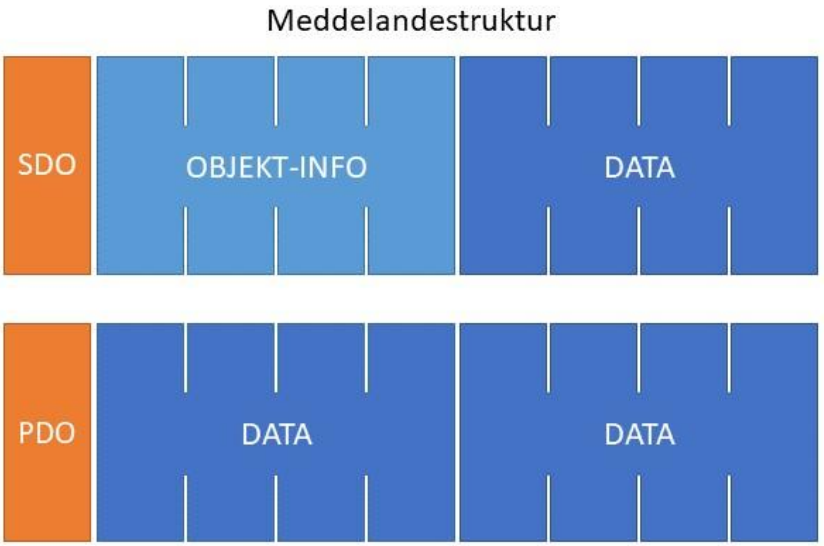
2.2 Jitter
Jitter är det som uppstår när ett tidsförlopp som ska upprepas med jämna intervaller inte sker stabilt. Det är oönskade variationer hos signalen som kan skapas av störningar eller noggrannhet hos hårdvaran. För realtidsapplikationer kan det vara förödande därför det skapar fördröjningar och dataförluster.

2.3 Fältbussystem
En fältbuss är en kommunikationsbuss som arbetar digitalt med informationsflöde mellan olika enheter. Det används för sammankoppling av styrsystem i nätverk. Det finns olika typer av standardkablar, och olika typer av protokoll såsom EtherCAT eller CANopen. Beroende på vilken typ av kabel som används kan termineringsmotstånd behövas. Det förhindrar att delar av signalen kan studsa tillbaka i kabeln och skapa störningar[2].

2.4 SDO och PDO
Service data object (SDO) och Process data object (PDO) är två sätt att skicka och ta emot data mellan enheter via fältbussystem[3][4]. En SDO kan använda sig av hela enhetens objektbibliotek, såsom: variabler, funktioner och inställningar. En PDO kräver att det objekt man vill skriva till redan är konfigurerat till en viss adress. Även om båda sätten uppfyller samma funktion kan de vara olika bra beroende på situationen.

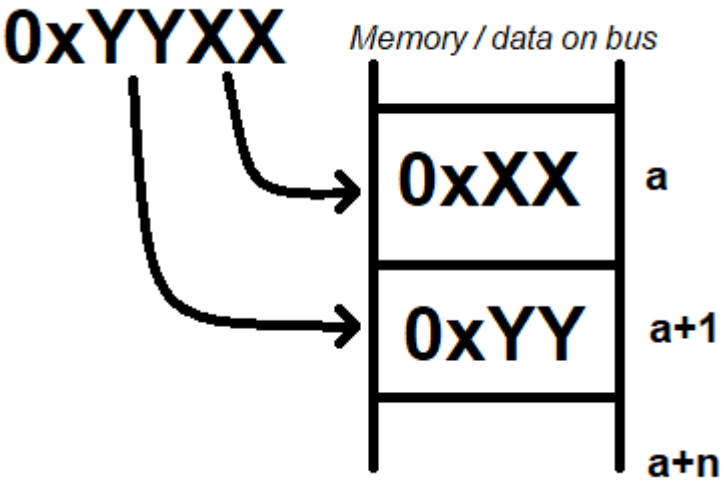
Meddelandet i en SDO kan bestå av 8 bytes. Den första byten signalerar hur mycket data som ska skickas eller tas emot, de andra tre byten bestämmer vilket objekt som ska skrivas till eller läsas, och de sista fyra byten är data. Storleken på datan kan variera beroende på vad för signal som skickas.

Eftersom en PDO redan är konfigurerad behöver inte lika stort meddelande skickas. Här skickas främst block med data, och därför kan mer data skickas samtidigt.
Skillnaden mellan SDO och PDO visualiseras i ett exempel i figur 2 nedan, där en PDO kan innehålla dubbelt så mycket data som en SDO, men ta upp lika mycket plats på fältbussen.



Figur 2: Meddelandestruktur för SDO och PDO

Både SDO och PDO använder sig av datastrukturen "little endian", vilket handlar om byteordningen[5]. Där skrivs den minst signifikanta byten först, och den mest signifikanta byten sist. Figur 3 visar hur den ordningen är uppbyggd mer visuellt, där talet 0xXXYY skrivs som 0xYYXX.



Figur 3: Little Endian byteskiftning

2.5 Ethernet for Control Automation Technology (EtherCAT)

EtherCAT är ett protokoll för Ethernet-baserade fältbussystem. Det togs fram av företaget Beckhoff där fokus var korta cykeltider, lågt jitter med noga synkronisering samt att kostnaden skulle minimeras. Protokollet är standardiserat enligt IEC 61158 och är användningsbar för realtid[6].

Ett nätverk kan bestå av flera noder där en av dem agerar som huvudenhet och de resterande som slavenheter. Huvudenheten kan skicka ett meddelande till hela nätverket och när meddelandet passerar en slavenhet skickar den det vidare, men kan välja att skicka med något samtidigt. Detta upprepas tills den sista noden i nätverket får meddelandet, och därefter skickas det tillbaka samma väg tills det når huvudenheten. På detta vis får alla noder ta del av samma information. Dock är det endast huvudenheten

som har möjlighet att aktivt skicka ut ett meddelande, medan alla andra noder endast vidarebefordrar det. Detta system förhindrar att flera saker skickas på nätverket samtidigt, och därmed onödiga fördröjningar[6].

EtherCAT-kortet som använts i projektet är av typen FC1121 av Beckhoff. Kortet används för att integrera en PC som en slavenhet i ett EtherCAT system. Kortet ansluts på PC:ns moderkort via PCIe[7].

2.6 CANopen

CANopen är ett standardiserat protokoll för kommunikation över CAN [8]. Det utvecklades som ett standardiserat inbyggt nätverk med fokus på flexibla konfigurationsmöjligheter. Idag används CANopen i ett flertal olika områden, bland annat medicinsk utrustning, automatiseringsprojekt och fordonsindustrin.

2.7 CANoe av Vector

CANoe är ett program skapat av Vector som kontrollerar och analyserar olika styrenheter. CANoe används inom fordonsbranschen vid testning av nya styrenheter[9], men kan även konfigurera användningen av EtherCAT och CANopen system.

Genom att sätta upp CANoe som huvudenhet kan EtherCAT slavenheter integreras och styras antingen manuellt eller genom att köra förskrivna program. Med ett importerat ESI bibliotek, av det EtherCAT kortet som används, möjliggörs konfigurering av kortet.

Det går även att skicka SDO:er och PDO:er över ett CAN-nätverk via CANoe. Genom att sätta upp egenskrivna .dbf filer som håller koll på alla adresser och minnesbitar förenklas tolkning och sändning av data på CAN-bussen. CANoe har ett plug-in för att skriva databasfiler av typen .dbf.

2.8 XL Driver Library av Vector

XL Driver Library är ett C-kod bibliotek med funktioner och definieringar för att kommunicera med en CAN-buss via ett Vector-interface. Dessa olika funktioner används för att själv bygga upp en egen applikation via C-kod. Det ger även möjligheten att implementera CAN kommunikation i andra applikationer.
XL Driver Library ger även möjligheten att bygga applikation för CAN FD, LIN, Ethernet, FlexRay, DAIO, ARINC[10].

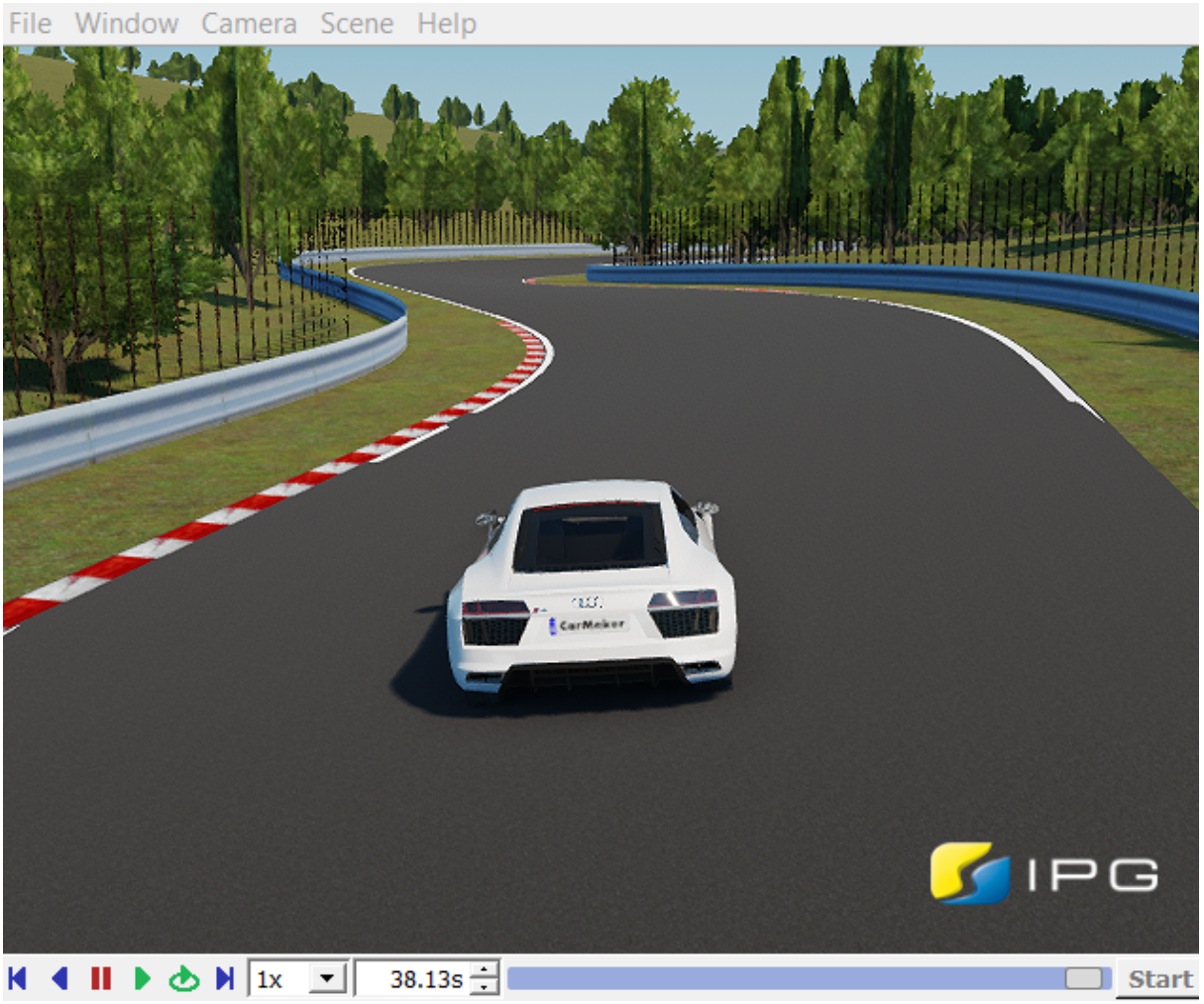
2.9 CarMaker av IPG Automotive

CarMaker är ett simuleringsverktyg för fordon framtaget av IPG Automotive. Det är en testplattform för att testa bilmodeller, mjukvara och hårdvara. För att göra utvecklingsfasen av nya bilmodeller så lätt som möjligt är CarMaker byggt för att kunna integreras med verktyg framtagna av andra företag.

CarMaker används av Volvo Cars för dess omfattande och möjlighet till att sätta parametrar till fordon, miljö och situationer i en simulering. Det är till exempel möjligt att ta fram data vid en exakt tidpunkt något inte gick som planerat.

Den version av CarMaker som projektet är anpassat för är version 10.1. Unikt med denna version är MovieNX som är en videoupplevelse av simuleringen, se figur 4. Tidigare versioner använder IPG Movie som inte klarar av att hålla samma grafik som MovieNX.

CarMaker ger även användare möjligheten att ändra och modifiera programmet. Via separata filer i programmets folder kan egna funktioner skrivas in och få programmet att köra som man önskar. Det går därigenom att sätta upp en kommunikation med andra applikationer, nätverk och drivrutiner, utan tredjepartsmjukvara.



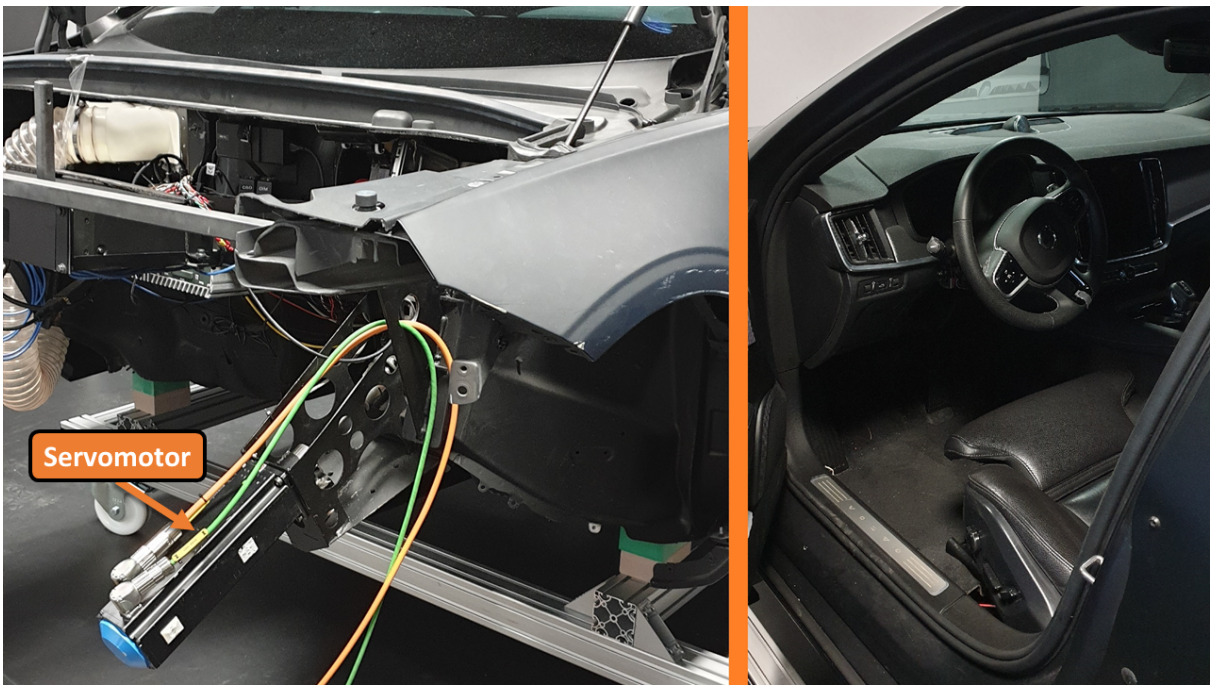
Figur 4: Simulering med CarMaker via MovieNX

2.10 Reglersystem och PD-Regulator

Reglersystem är ett begrepp som brukar användas till automatiserade system som på något sätt kan sköta sig själva utan mänsklig övervakning[2]. Det är vanligt att sådana system använder sig av någon slags regulator. Det finns olika varianter av regulatorer och PD-regulator är ett exempel. Den har två delar, förstärkning och dämpning. Den får ett börvärde som insignal, vilket den reglerar systemet utefter. Systemets utsignal och verkliga värde benämns ärvärde, vilket kan återkopplas tillsammans med insignalen för vidare reglering.

2.11 Styrenhet & Servomotor

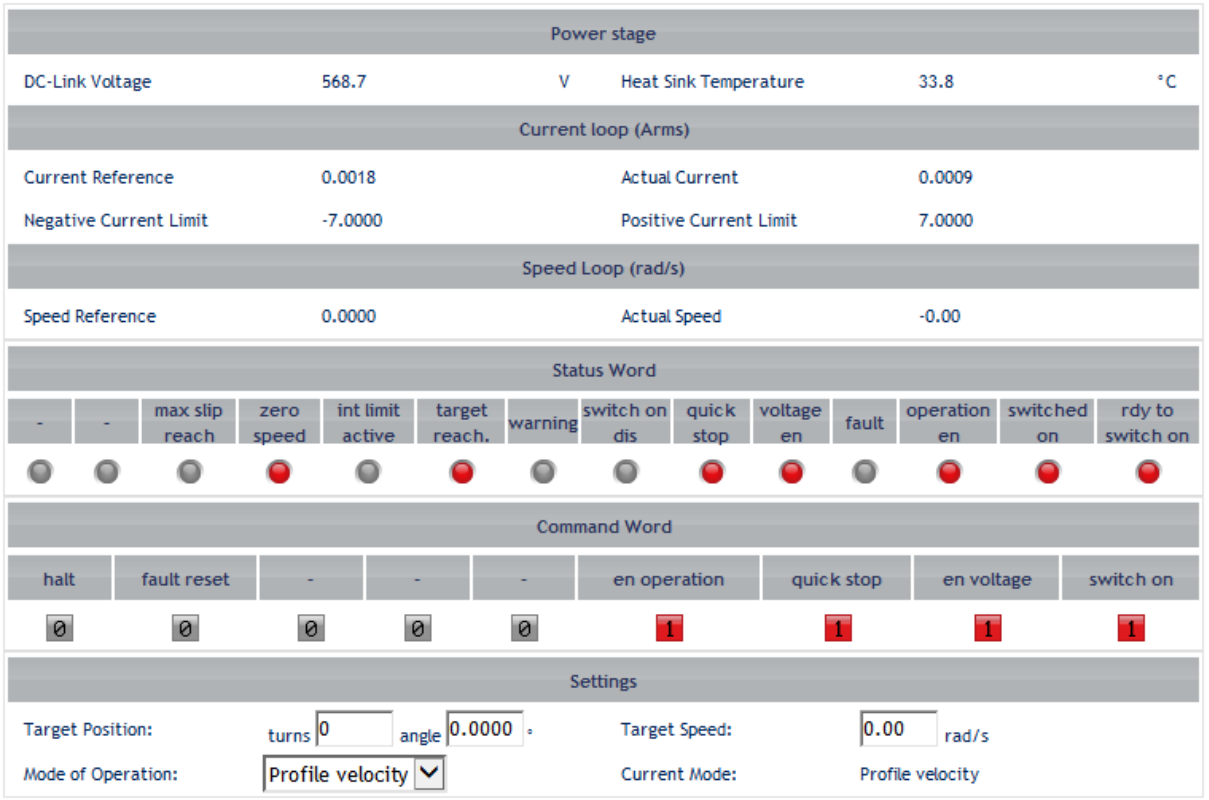
Styrenheten är en AxN Drive framtagen av Phase Motion Control[11]. Servomotorn är också framtagen av Phase Motion Control och är av typen Ultract 3N med nominellt moment på 13,1 Nm[12]. Styrenheten styr motorn med hjälp av en PD-regulator där en givare i motorn läser av en vinkelposition. Vinkelpositionen återkopplas till PD-regulatorn för reglering. I figur 5 visas servomotorn kopplad, via en rattstång, till en fysisk bilratt.



Figur 5: Servomotor vars axel är kopplad till en bilratt

2.12 Cockpit av Phase Motion Control

Cockpit är ett program skapat av Phase Motion Control för att enkelt kunna konfigurera deras styrenheter. Exempel på konfigurationer som kan göras är om styrenheten ska kommunicera via EtherCAT eller CAN och ändring av PDO:ers input och output. I figur 6 visas "Device Control" fönstret som kan användas för att bitvis sätta upp systemet så att det körs efter ens preferenser. Man kan styra det via vridmoment, vinkelhastighet eller position. Det går även ändra värden i PD-regulatorn för ökad prestanda[13].



Figur 6: Device Control fönster från programmet Cockpit.

2.13 Realtidsdator och Windows

En RTD är förutsägbar. Den kommer genomföra sina aktiviteter på en förväntad tid och i en förväntat ordning. Resultatet är förutsägbar och en RTD kan klassificeras som ett realtidssystem, vilket gör den deterministisk[1].
Hård-, fast- och mjuk-realtid är tre olika sätt att klassa ett realtidssystem:

- Hård: Systemet uppfyller alltid sina aktiviteter inom förväntat tidsintervall
- Fast: Systemets aktiviteter uppnår oftast inom sitt förväntat tidsintervall. När en aktivitet inte uppnår förväntat tidsintervall blir dess resultat oanvändbart. Det gör att en aktivitets process kan avbrytas om deadline överskrivs.
- Mjuk: Systemets aktiviteter försöker uppnås inom sitt förväntade tidsintervall. Användbarheten av resultatet, från aktiviteten, ökar eller minskar med hur stor tidsdifferensen är jämfört med det förväntade tidsintervallet. Till exempel ju längre tid som går efter en deadline, desto mindre värde har resultatet för systemet.

Windows är inte deterministisk och garanterar inte att en aktivitet blir klar inom den tänkta deadlinen. Det kan klassas som ett realtidssystem med mjuk-realtid.

2.14 National Instruments - HIL

National Instrument - HIL är Volvo Cars nuvarande hårdvara för att köra simuleringsprogrammet CarMaker och sedan styra servomotorn via styrenheten.
Det är en RTD där CarMaker-simuleringen är exakt med ekvidistanta tidssteg. NI - HIL kommunicerar med styrenheten via EtherCAT och har av Volvo Cars redan uppsatta inställningar. Därför går denna rapport inte in på dess konfigurationer och inställningar, utan tester kommer endast ske på systemet för jämförelse

2.15 QueryPerformanceCounter (QPC)

QPC är funktioner i Windows för att beräkna tid. QPC baseras på en självständig räknare i systemet med upplösning som är mindre än 1µs. Den är fristående från processorn och har en fast frekvens från uppstart. Den kan därför användas för att beräkna korta tidsintervall[14]. I C-programmering ges den åtkomst via <windows.h>.
Frekvensen styrs oftast av en kvartsocillator, vilket vibrerar och ger ifrån sig signaler i en konstant frekvens. Vid tillverkning av dessa anger tillverkaren inom vilken tolerans frekvensen ges, ofta i storleksordningen miljondel, (ppm - parts per million).
Med en exempeltoleransen på +/- 10 ppm, vid en frekvens 1 000 000 Hz kan frekvensen i verkligheten vara mellan 1 000 010 Hz och 999 990 Hz. Detta ger i omräkning en felmarginal mot verklig tid som främst blir påtaglig vid längre tidsperioder. Vid tidsintervallet 1 ms är differensen mot verklig tid +/- 10 ns och
vid tidsintervallet 1 vecka är differensen mot verkliga tid +/- 6.08 s.
I detta projekt anses differensen vara så liten att den bortses.

3. Metod

För att kunna övergå från Volvo Cars existerande RTD omkopplas styrenheten till PC:n via EtherCAT- och CAN-buss. Hela systemet kan ses som två delsystem, en virtuell bil och en fysik ratt. Delsystemen behöver kunna kommunicera med varandra utan tidsfördröjning och utan för mycket jitter. Kommunikationen för systemet kan sättas upp på olika sätt. För att jämföra dessa måste passande tester tas fram. Därför går det att dela upp arbetet i fyra delar: Uppsättning av systemet, stabilisering av CarMaker, uppbyggnad av en egen applikation via CAN och specificering samt genomförande av tester.

3.1 Uppsättning av system

För att koppla samman den fysiska simulatorn med simuleringsprogrammet krävs det bakgrundskunskap om de verktyg som används. Det är viktigt att förstå hur de kan jobba med varandra och hur informationen kan skickas mellan dem. Kunskap inhämtas genom att läsa igenom tillgängliga manualer, samt genom genomgångar och dialog med de företag som tagit fram respektive programvara. Vidare testas olika sätt att styra och konfigurera styrenheten. För att nå slutmålet testas konfigureringar i olika steg listade här:

- Styrning via Cockpit.
- Styrning via CANoe över EtherCAT.
- Simulering i CarMaker via CANoe över EtherCAT.
- Styrning via CANoe över CAN.
- Simulering i CarMaker via CANoe över CAN.

3.2 CarMaker stabilisering

CarMaker är ett program som är byggt för en RTD, och räknar med att hårdvaran har en exakt prioriteringsordning på all exekvering och kommer nå de deadlines som är nödvändiga för att göra simuleringen verklighetstrogen. Cykeltiden ska vara 1 ms[15] där indata, beräkningar och utdata ska hinna genomföras. På en standard Windows PC uppfylls inte dessa krav. Windows har möjligheten och kör CarMaker snabbare än den önskade periodtiden, men kompenserar sedan genom att exekvera kommande perioder långsammare för att medeltiden ska bli korrekt. Därför behöver CarMakers programkod stabiliseras med ekvidistanta tidssteg som ser till att programmet körs med den cykeltid som eftersträvas.

Först används olika metoder för att stabilisera CarMaker till förväntade värden. Sedan väljs och finjusteras den stabiliseringsmetod som givit närmast mätresultat för cykeltiden 1 ms.

3.3 CarMaker med CAN

För att möjliggöra en egen CarMaker applikation som skickar och tar emot signaler via CAN-bussen skapas en egen fil med syftet att sköta den kommunikationen. Genom att använda Vectors XL Driver Library finns alla basfunktioner redan färdiga, och det blir smidigt att bygga större och mer avancerade funktioner som kan implementeras i CarMaker.

3.4 Genomförande av tester för uppbyggda system

Det totala systemet kan fördelas i tre enheter som visas i figur 7: PC/HIL för simulering, styrenhet och servomotor. CarMakers simuleringscykeltid är 1 ms, vilket hela systemet får utgå ifrån. Idealet är att CarMaker varje millisekund skickar ett börvärde till styrenheten och läser in ett uppdaterat momentvärde varje millisekund. Det innebär att styrenheten också bör skicka och ta emot data varje millisekund. Det samma gäller för

styrning av motorn och avläsning av dess givare. Om alla enheter opererar efter idealet 1 ms så ger det ett realtidssystem utan dataförluster. Systemet blir förutsägbart och i förlängningen möjliggör det att den verkliga ratten stämmer överens med den simulerade ratten.

Figur 7: Informationsflöde i systemet

Med föregående stycke som grund och genom diskussion med anställd på Volvo Cars bestäms det att tre uppkopplade system bör testas. Den färdiga HIL-versionen, EtherCAT-versionen samt CAN-versionen. Även vilka typer av tester bestäms för att jämföra prestandaskillnader. Testerna ger data om:

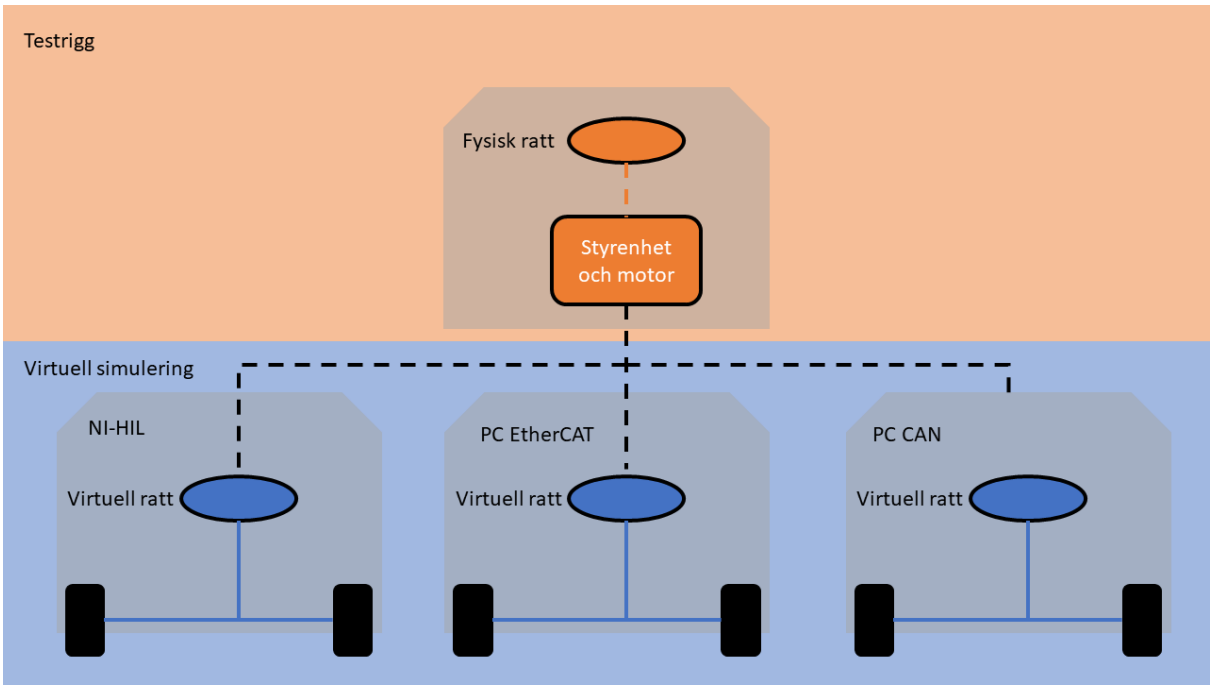
- Cykeltiden som CarMaker exekveras i, och således cykeltiden på när in- och utdata läses in respektive skickas.
- Cykeltiden vid överbelastning. Eftersom ett problem med Windows är att det inte finns någon exakt prioriteringsordning för exekvering, är det viktigt att ta hänsyn till att program behöver vara aktiva i bakgrunden. Program kan även startas på grund av att Windows behöver köra schemalagda processer, exempelvis antivirusprogram.
- Fördröjningstiden. Tiden det tar att skicka en signal från CarMaker till styrenheten, till att styrenheten skickat tillbaka den och att CarMaker uppfattar att signalen är tillbaka.
- Teoretisk och praktisk inställning av PD-regulatorn. CarMaker har en simulerad dämpning och styvhet inbyggd, och det är viktigt att styrenhetens PD-regulator inte har för stor effekt på hela systemet. En matematisk formel för att beräkna styvheten är därför nödvändigt och sedan praktisk testning för att se vilka värden som fungerar bäst. Detta testas endast på ett system, eftersom det ska vara samma för alla.
- Förstärkning och fasförskjutning av systemet mellan förare och simulering för att utvärdera olikheten mellan verklig ratt och simulerad ratt vid mänsklig interaktion.

4. Uppsättning system

I detta kapitel beskrivs hur uppkoppling av systemet görs, samt olika sätt att styra och koppla samman simuleringen med den fysiska hårdvaran. Hårdvaran som används för att koppla samman de olika lösningar är specificerade i bilaga 2. Här följer några av de viktigaste:

- Servomotor med en axel kopplad till en bilratt
- Styrenhet för styrning av servomotor med kommunikationsmöjligheter över olika kommunikationsprotokoll som CAN och EtherCAT.
- NI - HIL för simulering av CarMaker med kommunikationsmöjligheter över till exempel EtherCAT.
- PC för simulering av CarMaker med kommunikationsmöjligheter över till exempel EtherCAT och CAN.

I figur 8 illustreras hur olika system kopplas samman med testriggen. Där de blå virtuella simuleringsenheterna endast kan kopplas in var för sig. Testriggen består av en fysisk ratt med styrenhet och servomotor som kan ses som en förlängd rattstång adderad på den simulerade ratten.



Figur 8: Illustrering av olika konfigurationer av systemen

4.1 Styrning via Cockpit

För att styra och läsa av styrenheten via Cockpit är en seriell kabel ansluten mellan PC:n och styrenheten. Cockpit arbetar mycket på bitnivå. Genom att slå av och på olika signaler kan motorn styras på olika sätt. Styrenheten sätts i körläget för styrning på vinkelhastighet. Det finns även två andra alternativ av styrning, och de är position eller moment. När strömmen är påslagen och styrenheten sätts i körläge får servomotorn ström och ratten blir styv. Om ett moment läggs på ratten vill den återgå till sin ursprungliga position, eftersom vinkelhastighetens börvärde på ratten är ställd till noll. Genom att skicka in ett annat börvärde, med enheten rad/s, börjar ratten rotera. Den roterar medsols respektive motsols beroende på om det är ett positivt eller negativt värde som skickas. Hur hårt motorn jobbar fås genom att läsa av strömmen som används för att rotera ratten, som ges i enheten A/rad. Genom att ändra konstanter i den inbyggda PD-regulatorn kan

olika styvheter uppnås.

Cockpit kan ställas in mer komplext, och har mycket bakomliggande funktioner som kan användas. Men för att kunna få ett slutet system är rattens vinkelhastighet som börvärde och momentet som ärvärde tillräckligt.

4.2 Styrning via CANoe (EtherCAT)

För att styra testriggen via EtherCAT används CANoe för att testa anslutningen. Detta genomförs fysiskt genom att dra en Ethernet kabel mellan styrenheten och PC:n. De drivrutiner som krävs laddas ner via Vectors hemsida.[16]. I Cockpit går det exportera en .xml-fil som importeras till CANoe:s ESI bibliotek för att kategorisera och namnge möjliga PDO:er och andra inställningar för styrenheten. Därefter går det att söka efter tillgängliga EtherCAT slavenheter. När styrenheten hittas, skapas en simulerad huvudenhet av CANoe som går att styra i programmet. För att testa anslutningen mellan styrenheten och CANoe läggs de önskade variabler in i PDO konfigurationen. Därefter kan deras värden ställas in och samma resultat uppnås som i kapitel 4.1.

4.3 Simulering med CarMaker via CANoe (EtherCAT)

För att testa anslutningen av hela systemet behöver CarMaker kopplas samman med CANoe. Det görs genom att starta upp CarMaker, och läsa in de variabler som man vill få tillgång till i CANoe. Därefter skapas en ny nod med en CAPL-fil i CANoe. Där skrivs kod, för att koppla samman CarMaker och CANoe och de variablerna från de olika systemen[16][17]. CarMakers vinkelhastighet på ratten kopplas samman och skickas till styrenhetens vinkelhastighet. Styrenhetens moment kopplas samman och skickas till CarMakers moment. På så sätt blir det en sluten loop. En omvandling behövs mellan enheterna, så att värdena är i rätt enhet, för CarMaker, respektive styrenheten. Dessa omvandlingsformler finns i Phase manualen[15]. När variablerna är sammanlänkade går det styra simuleringens bil med den fysiska ratten i testriggen.

4.4 Styrning via CANoe (CAN)

För att styra testriggen via CAN används CANoe för att testa anslutningen. Fysiskt krävs det att en CAN-buss kopplas upp med ett Vector-interface som går in i datorn. Det krävs även ett termineringsmotstånd på var sin ända av CAN-bussen för att motverka störningar.

För att testa anslutningen skickas SDO:er från CANoe till styrenheten. Först måste initieringsmeddelanden skickas för att aktivera styrenheten. Sedan går det manuellt att skicka data till de objekt man vill skriva till. För att läsa av något objekt skickas även SDO:er manuellt.

Det går att använda sig av PDO:er istället för SDO:er för att få objekt att skickas automatiskt. I styrenhetens PDO-konfiguration aktiveras de objekt man vill läsa av eller skriva till. De sätts även till att skickas cykliskt, och efter initieringsmeddelandet kommer CANoe kunna läsa av det.

4.5 Simulering med CarMaker via CANoe (CAN)

Genom att skapa en dbf-fil i CANoe kategoriseras och tolkas data som skickas på bussen till specifika variabler och/eller signaler. Detta blir mycket viktigt för att kunna koppla samman CarMakers variabler med styrenhetens. I dbf-filen går det även lägga till en faktor på variablerna eftersom de måste omvandlas från rådata till enhets mätvärden eller tvärtom. I CANoe:s kopplas de samman med CarMakers respektive variabler. Innan

simuleringen kan köras skickas några SDO:er manuellt för att aktivera bussen och starta systemet. Dessa är bland annat startkommando till styrenheten, vinkelhastighet som val av körläge och aktivering av servomotorn. Därefter går det styra simuleringens bil med den fysiska ratten i testriggen.

5. Lösning stabilisering av CarMaker

Genom ändring av CarMakers kod i filen “User.c” ges olika resultat på CarMakers cykeltid. En cykeltid på 1 ms eftersöks och olika metoder implementeras i en funktion som ligger i början av CarMakers simuleringscykel [15]. Där den bästa lösning eftersträvas, för att senare jämföras med RTD.

Ändringar som genomförs:

1. Ingen ändring: Koden justeras inte alls.
2. Användning av QPC: Två funktioner används, QueryPerformanceCounter() för att få fram antal klockcykler räknaren genomfört och QueryPerformanceFrequency() för att få räknarens frekvens. Delas antalet klockcykler med frekvensen fås tiden räknaren har varit aktiv. Genom att beräkna differensen mellan en starttid och en sluttid kan en önskad verklig tidsdifferens beräknas. Vilket i sin tur kan användas för att hålla programmet kvar tills en tidsdifferens på 1 ms uppnåtts.
3. Användning av Sleep() funktion: Windowsfunktionen “Sleep(1)” används och får processorn/tråden att stoppa i 1 ms.

De versioner av CarMaker som projektet går vidare med är utan justeringar, samt med användning av QPC. Funktionen “Sleep()” presterade vid enklare tester sämre än QPC. Vilket föranledde att inga djupare och dokumenterade tester genomförs.

6. Lösning egen applikation i CarMaker

Genom att skapa en egen header-fil byggd med XL Driver Library som grund, ges möjligheten att skapa funktioner som sköter all initiering av CAN-bussen och hantering av all trafik via ett Vector-interface. Header-filen byggs för att senare implementeras i CarMaker.

CarMaker är strukturerat för att användare ska ha möjligheten att kunna skriva sin egen kod och kunna använda det för egna skräddarsydda simuleringar. Därför är det byggt med en stor main-fil som sköter hela simuleringen och hämtar alla sina funktioner från andra filer. En av filerna som den hämtar funktioner ifrån är "User.c" som främst är gjort för att användare ska kunna ändra i koden utan att behöva ändra saker i main-filen. Det minimerar risken att programmet blir oanvändbart vid programmeringsfel, eftersom den bakomliggande strukturen av programmet är den samma.

"User.c" innehåller 29 olika funktioner som alla kan fylla viktiga syften. De kan läggas in i fem kategorier, vilka är initiering, start av simulering, slut av simulering, simuleringscykel och nedstängning av programmet. För att vår applikation ska fungera som önskat krävs det att alla kategorier ändras. Däremot är inte alla 29 funktioner nödvändiga för att nå målet.

6.1 Initiering av CAN-buss

En header-fil byggs på XL Driver Library som initierar CAN-bussen och hanterar all trafik. Sex större funktioner skapas som senare inkluderas i CarMaker. Nedan följer beskrivning på dessa:

- Funktion för initiering av CAN: Först laddas de drivrutiner för det interface som används in till programmet. Vidare undersöks om det finns några tillgängliga kanaler, samt ifall de har CAN-support. Portar öppnas med hänsyn till vilket CAN-interface som används. När det är avklarat, sätts busshastigheten till 1Mbit/s. Om funktionen inte skulle hitta drivrutiner, tillgängliga kanaler eller inte lyckas att öppna porten, kommer ett felmeddelande att returneras. Vid lyckad initiering returneras 0.

- Funktion för nedstängning: Först undersöks om det finns någon öppen port, och i sådana fall stänger den. Sedan stänger den ner de drivrutiner som är öppna så att programmet inte kan kommunicera. Sist returneras alltid 0.

- Funktion för felsökning: Alla kanaler och deras namn skrivs ut i ett konsolfönster. Ingenting returneras.

- Funktion för att få kontakt med bussen: Gör att inkommande meddelanden aviseras till programmet.

- Funktion för att läsa bussen: Först inväntas att ett meddelande på bussen ska komma. När något har ankommit kontrollerar den ifall sändarens adress är från styrenheten och sparar sedan meddelandet, annars ignoreras meddelandet. Vid önskat meddelande sparas 32 bitar, via little endian, big endian, till en global variabel. Den globala variabeln går sedan att nyttja i "User.c". Sist returneras 0.

- Funktion för att skriva på bussen: Denna funktion skickar meddelanden ut på bussen. Därför krävs information om vad som ska skickas och vart den ska skickas: Adressen dit meddelandet ska skickas, vilken kanal det ska skickas till, storleken på meddelandet givet i antal bytes, meddelandet givet hexadecimalt och en signal om

talet måste skiftas bytevis eller ej. Med all den indata lägger funktionen in det i ett "event", och kontrollerar om porten är öppen och vilken kanal som det ska skickas på. Sist returneras 0 om det gick att skicka, och större än 0 om det inte gick, där det returnerade värdet är en felkod beroende på vad som var problemet.

6.2 CAN i CarMaker

För att implementera CAN i CarMaker läggs kod till i "User.c", där bland annat den tidigare skapade header-filen inkluderas. Genom ändring i följande delar av koden möjliggörs kommunikation över CAN-bussen för CarMaker. Applikationen kan därefter skriva och läsa data, till och från styrenheten.

- Funktion för initiering: Detta steg sker när CarMaker startas. Här initieras CAN-bussen med funktioner från XL Library och den egna header-filen, så att applikationen kan läsa och skriva till bussen.

- Funktion vid start av simulering: PDO:er skickas på bussen för att aktivera styrenheten, samt sätter den i körläge vilket möjliggör styrning av servomotorn. Det skickas även ett meddelande för att ta bort eventuella felmeddelande som är aktiva, vilka kan hindra styrenheten från att köra servomotorn.

- Funktion vid slut av simulering: PDO:er skickas på bussen för att nollställa vinkelhastighetens börvärde och inaktiverar körläget så att servomotorn inte kan köra.

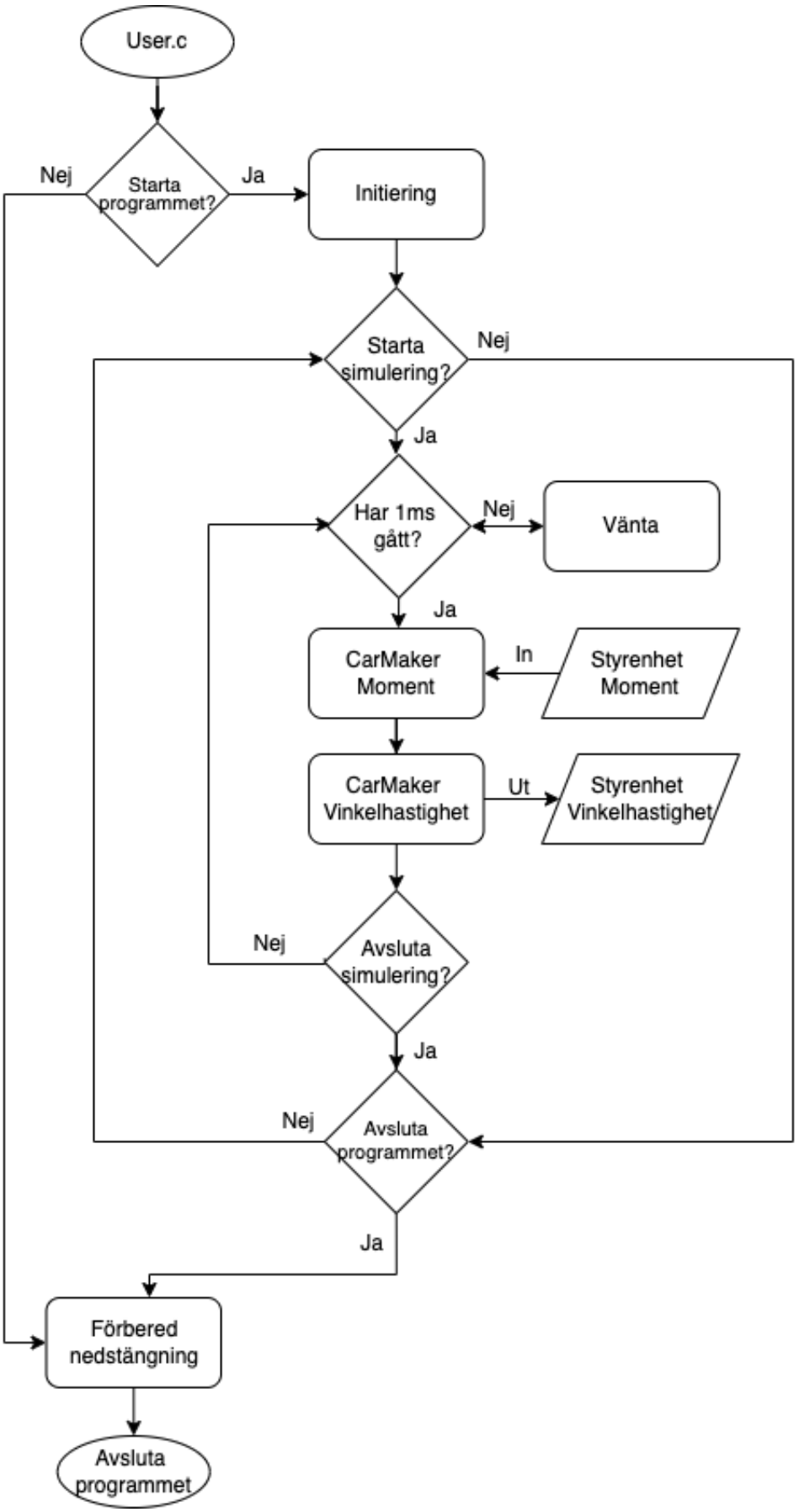
- Funktion i början av simuleringscykel: Denna funktion kallas i början av varje cykel som CarMaker kör. Här säkerställs att det har gått minst 1 ms sedan senaste gången funktionen exekverades, (enligt kap. 5). Den säkerställer därmed att cykeltiden för applikationen blir ~1 ms.

- Funktion för beräkningar: Moment som lästs in från bussen skrivs in i CarMakers variabel för styrning, vilket används för simuleringens beräkningar.

- Funktion för utdata: PDO skickas på bussen med vinkelhastighetens börvärde. Genomförs endast var 4:e cykel, dvs var 4 ms. Detta eftersom styrenheten inte kan bearbeta data från bussen snabbare.

- Funktion för nedstängning: Funktion som stänger ner det som initierades när CarMaker startades.

I figur 9 visas hur "User.c" exekveras. Där framgår det att initiering och avslut enbart sker en gång vardera och att det är i simuleringsdelen som applikationen främst arbetar. För varje cykel läser programmet in ett momentvärde. Dock skickar styrenheten nytt momentvärde var 2-3 ms. Det gör att samma värde kommer läsas in under 2-3 cykler. Utsignalen är också begränsad till att skicka var fjärde cykel.



Figur 9: Flödesschema av CarMaker efter omskrivning av kod

7. Test och resultat

För att kunna avgöra om det är möjligt att övergå till att köra simuleringar via en PC genomförs tester. Både EtherCAT och CAN jämförs med den RTD som anses vara den korrekta lösningen. EtherCAT och CAN testas både med och utan stabilisering av cykeltiden. Totalt genomförs 5 olika test med 1-5 olika lösningar. All loggning av data sker i CarMaker och exporteras till Matlab för analys. Loggningen sker i slutet av varje simuleringscykel. Grafer och tabeller för alla tester kan hittas i bilaga 1.

7.1 Test 1, Jitter av CarMaker-loopen

Det första testet analyserar hur nära 1 ms CarMakers cykeltid exekveras. Detta mäts genom en tidsmarkör för varje gång som CarMaker når slutet av en cykel. Om allt fungerar perfekt skall CarMaker skicka ett meddelande varje 1 ms. Testet genomförs på banan “Racetrack_Nurburgring” utan en extern förare. Simuleringen körs i 60 s, efter stopp sparas loggad data.

Test 1 genomförs på följande konfigurationer:

1. CarMaker HIL på National Instruments (facit/utgångsläge)
2. CarMaker Windows med stabiliserad cykeltid via CANoe över EtherCAT
3. CarMaker Windows utan stabiliserad cykeltid via CANoe över EtherCAT
4. CarMaker Windows med stabiliserad cykeltid över CAN (XL Driver Library)
5. CarMaker Windows utan stabiliserad cykeltid över CAN (XL Driver Library)

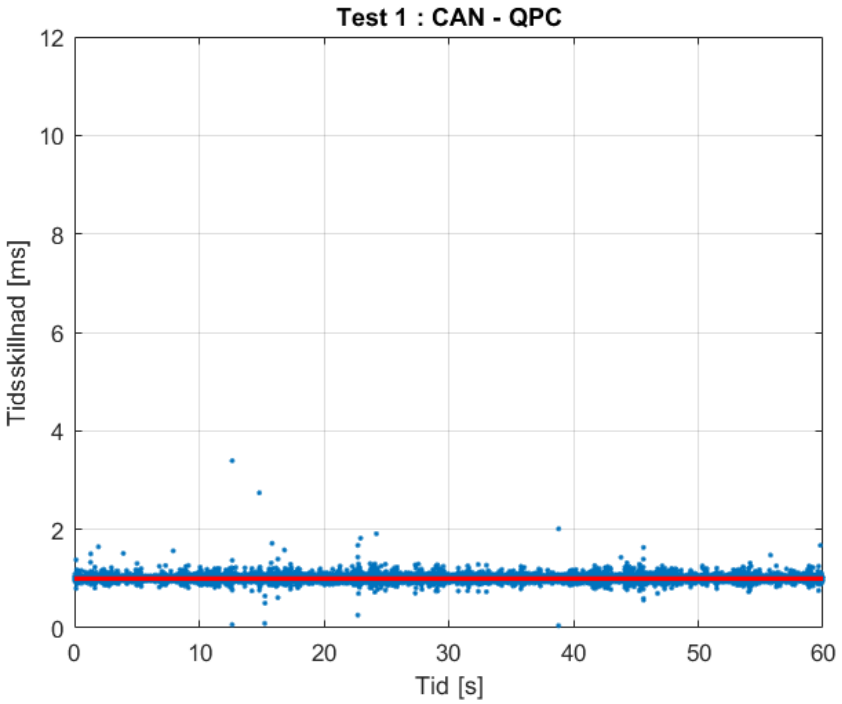
Test 1 ger följande information:

1. Plott över tidsspannet mellan varje cykel
2. Stapeldiagram över tidsspannet mellan varje cykel
3. Medelspridning över tidsspannet mellan varje cykel
4. Medeltiden över tidsspannet mellan varje cykel
5. Standardavvikelsen över tidsspannet mellan varje cykel

7.1.1 Test 1, Resultat

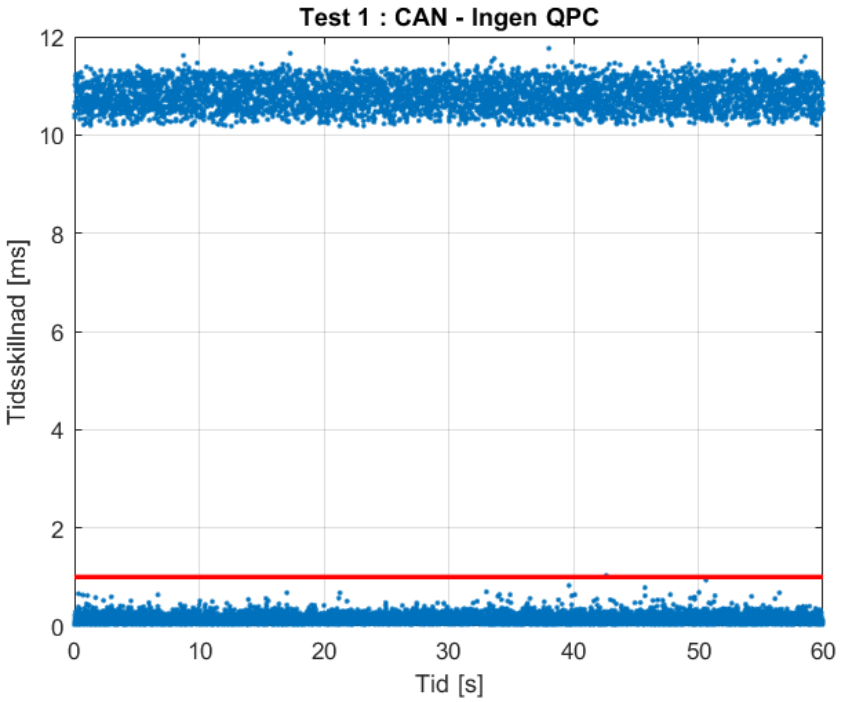
Test 1 visar att CarMaker har mindre jitter efter justeringar av koden. Den presterar inte lika bra som HIL men påvisar en stor förbättring utan någon justering alls av koden.

Figur 10 visar tidsdifferensen mellan varje cykel där CAN med stabiliserad cykeltid analyseras. Den röda linjen vid Y=1 är HIL som ligger exakt på 1 ms. Alla blå prickar visar tidsskillnaden mellan cykler för CarMaker simulerat med CAN. 1 ms är det värdet som eftersträvas, och nästan alla uppdateringar ligger stabilt runt det området. Några enstaka uppdateringar går uppemot 2-4 ms, men under simuleringen kördes 60 000 cykler, där en klar majoritet ligger mellan 0-2 ms.



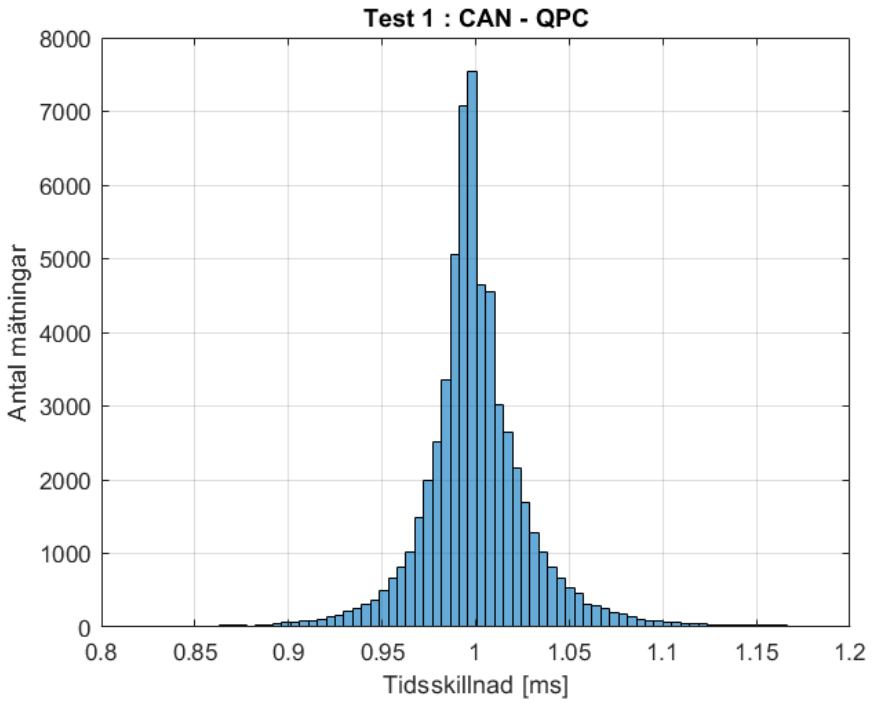
Figur 10: Tidsdifferens CAN - QPC

Figur 11 visar tidsdifferensen mellan varje cykel där CAN utan stabiliserad cykeltid analyseras. Den röda linjen vid Y=1 är HIL som ligger exakt på 1 ms. Alla blå prickar visar tidsskillnaden mellan cykler för CarMaker kört med CAN. Majoriteten av alla cykeltider ligger under 1 ms. För att kompensera så att medelvärdet av alla cykeltider blir 1 ms ökar Windows cykeltiden till 10-12 ms. Detta blir problematiskt när ingen av alla 60 000 cykeltider ligger på önskvärda 1 ms. När cykeltiden är hög innebär det att flera signaler som skickas från styrenheten inte kommer läsas in i CarMaker, och när den är låg kommer CarMaker att göra flera beräkningar på samma momentvärde från styrenheten, vilket ger dataförluster i realtidssystemet.



Figur 11: Tidsdifferens CAN - Ingen QPC

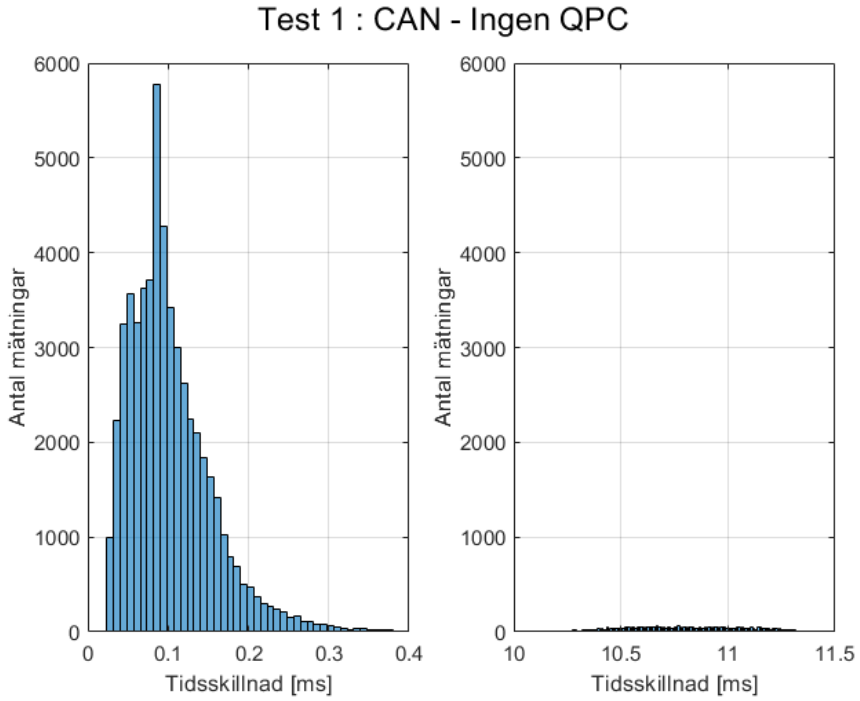
Figur 12 visar histogram över fördelningen av tidsdifferenserna för CAN med stabiliserad cykeltid. Det syns tydligt att majoriteten av alla värden ligger mellan 0,95-1,05 ms.



Figur 12: Histogram CAN - QPC

Figur 13 visar histogram över fördelningen av tidsdifferenserna för CAN utan stabiliserad cykeltid. En klar majoritet ligger under 0,4 ms, där de flesta som uppmätts ligger vid 0,1 ms. Men det finns även ett visst antal vid 10-11,5 ms som får programmet

att köras med medeltiden ~1 ms.



Figur 13: Histogram CAN - Ingen QPC

Tabell 1 visar en sammanställning av medelspridning, medeltid och standardavvikelse. De påvisar att CAN med stabiliserad cykeltid presterar närmast HIL i form av standardavvikelse och medelspridning. Däremot har CAN utan stabiliserad cykeltid det medelvärde som ligger närmast HIL-versionen, men medelspridningen och standardavvikelsen är så mycket sämre att stabiliserad cykeltid ses som ett måste för att inte få för stora dataförluster.

Tabell 1: Jämförelse av cykeltid

	Medelspridning [ms]	Medeltid [ms]	Standardavvikelse [ms]
HIL	$8{,}18 \cdot 10^{-24}$	1	$2{,}998 \cdot 10^{-12}$
EtherCAT - QPC	0,0579	1,002	0,2406
EtherCAT	8,4929	0,9992	2,9143
CAN - QPC	0,0463	1,0018	0,2152
CAN	8,8197	0,9999	2,9698

7.2 Test 2, Stresstest

Det andra testet analyserar hur systemet fungerar när Windows överbelastas. Detta mäts på samma sätt som i test 1 och genomförs på samma bana, utan förare i 60 s. Skillnaden blir att vid 10 s startas Google Chrome, vid 20 s startas Outlook och vid 35 s startas Matlab. Programmen startas manuellt. Anledningen till att just dessa tre program används är för att de kräver olika mycket av Windows processor. Test 2 testas inte med HIL-versionen på National Instruments därför att en RTD har en prioriteringsordning så att det inte ska gå att störa systemet.

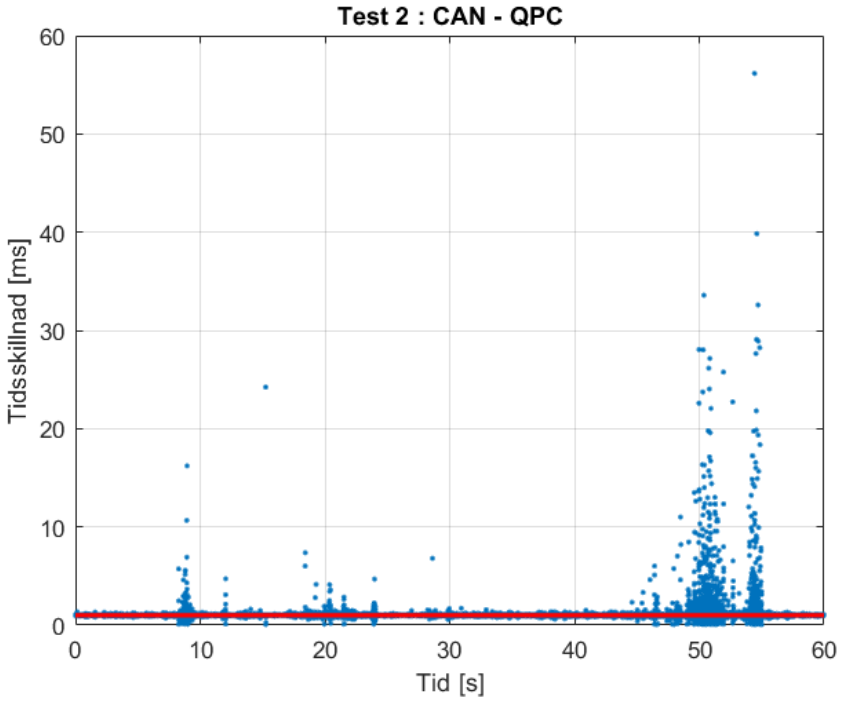
Test 2 genomföras på följande konfigurationer:
1. CarMaker Windows med stabiliserad cykeltid via CANoe över EtherCAT
2. CarMaker Windows utan stabiliserad cykeltid via CANoe över EtherCAT
3. CarMaker Windows med stabiliserad cykeltid över CAN (XL Driver Library)
4. CarMaker Windows utan stabiliserad cykeltid över CAN (XL Driver Library)

Test 2 ger oss följande information:
1. Plott över tidsspannet mellan varje cykel
2. Stapeldiagram över tidsspannet mellan varje cykel
3. Medel Spridning över tidsspannet mellan varje cykel
4. Medeltiden över tidsspannet mellan varje cykel
5. Standardavvikelsen över tidsspannet mellan varje cykel

7.2.1 Test 2, Resultat

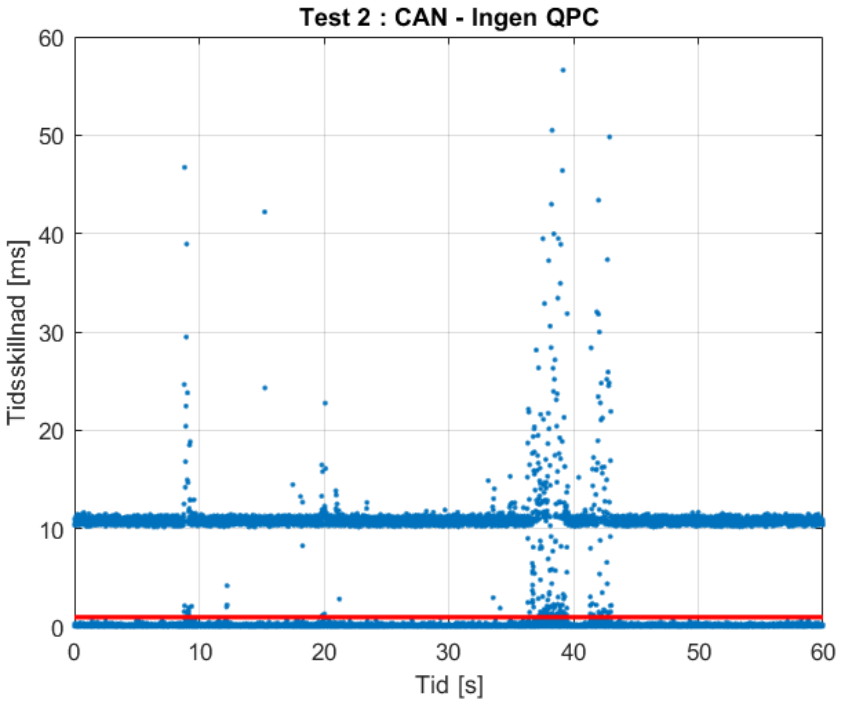
Test 2 visar att både EtherCAT och CAN-applikationen påverkas när Windows överbelastas. Skillnaden för tidsdifferensen, när mindre krävande program startas, märks av men är fortfarande låg. När tyngre program startas tar det upp mycket av processorns kapacitet och simuleringen försämras. När de väl har startat och endast är igång i bakgrunden stör de applikationen minimalt.

Figur 14 visar tidsdifferensen mellan varje cykel där CAN med stabiliserad cykeltid analyseras. Den röda linjen vid Y=1 är HIL som ligger exakt på 1 ms. Alla blå prickar visar tidsskillnaden mellan cykler för CarMaker kört med CAN. Vid 10 s och 20 s syns det att Google Chrome och Outlook har startats, men det ligger fortfarande relativt stabilt runt 1 ms. Vid 35 s när Matlab startas tar det ett tag innan programmet kommer igång och simuleringen går fortfarande stabilt. Men efter ca 45 s syns det mycket tydligare att simuleringen inte klarar av att köra stabilt. Däremot märks det av att i slutet av simuleringen när Matlab redan har startats blir simuleringen stabil igen.



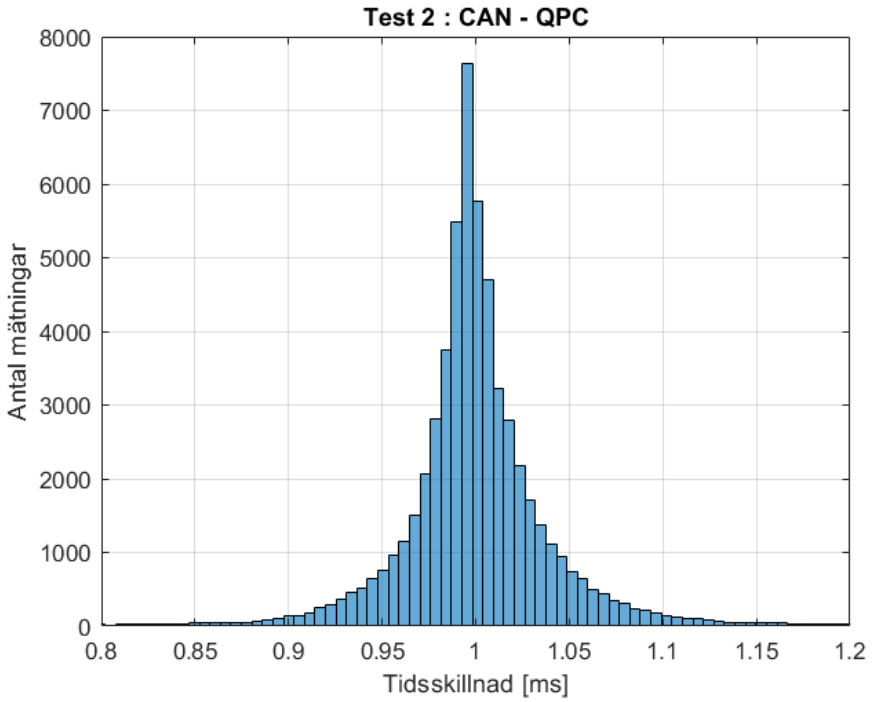
Figur 14: Tidsdifferens CAN - QPC

Figur 15 visar tidsdifferensen mellan varje cykel där CAN utan stabiliserad cykeltid analyseras. Den röda linjen vid Y=1 är HIL som ligger exakt på 1 ms. Alla blå prickar visar tidsskillnaden mellan cykler för CarMaker kört med CAN. Jämfört med figur 14 ser det ut som att det inte påverkas lika mycket utan stabiliserad cykeltid vid stresstest, men det ligger fortfarande väldigt långt ifrån 1 ms som eftersträvas.



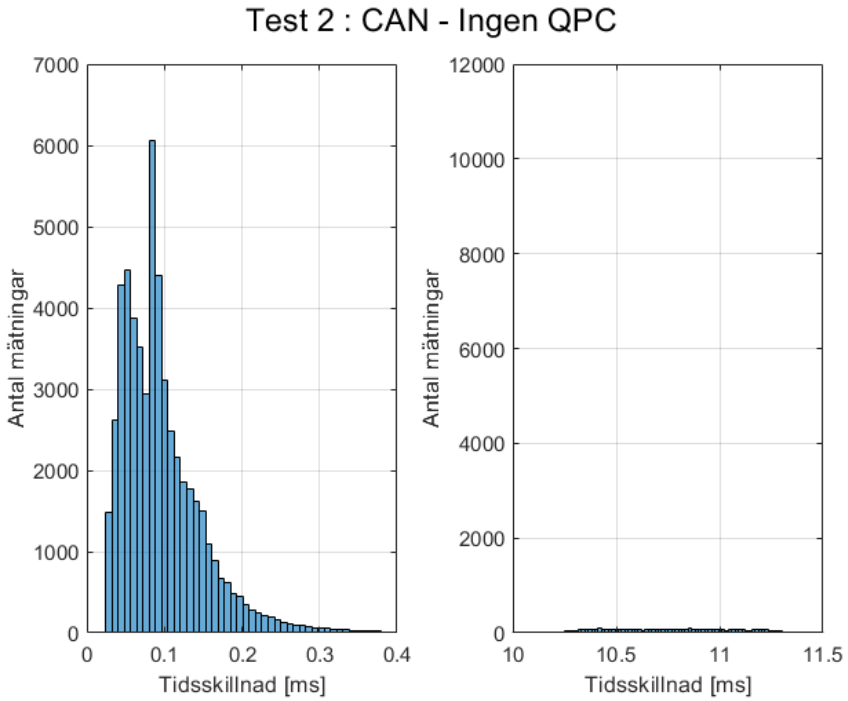
Figur 15: Tidsdifferens CAN - Ingen QPC

Figur 16 visar histogram över fördelningen av tidsdifferenserna för CAN med stabiliserad cykeltid. Det syns tydligt att majoriteten av alla värden ligger mellan 0,95-1,05 ms, men jämfört med figur 12 ser man en större spridning med värden som inte är exakt 1 ms.



Figur 16: Histogram CAN - QPC

Figur 17 visar histogram över fördelningen av tidsdifferenserna för CAN utan stabiliserad cykeltid. Det är ingen stor skillnad jämfört med figur 13.



Figur 17: Histogram CAN - Ingen QPC

Tabell 2 visar skillnader om man kör CarMaker med eller utan stabiliserad cykeltid vid stress, samt skillnaden mellan EtherCAT och CAN. Medeltiden påvisar att "utan stabiliserad cykeltid" presterar bättre. Men genom att studera medelspridningen så presterar "med stabiliserad cykeltid" över tiofaldigt bättre, vilket gör det till det bättre

valet. Jämförs test 2 med test 1 visar det vikten av att inte köra andra processorkrävande applikationer på Windows när simulering körs. Därför detta föranleder perioder med långa cykeltider, med dataförluster som följd. Fortsättningsvis kommer endast CAN och EtherCAT med stabiliserad cykeltid testas, för att det av test 1 och 2 anses vara det bättre alternativet.

Tabell 2: Jämförelse av cykeltid vid överbelastning

	Medelspridning [ms]	Medeltid [ms]	Standardavvikelse [ms]
EtherCAT - QPC	0,6903	1,0396	0,8308
EtherCAT	10,6414	1,0074	3,2621
CAN - QPC	0,5426	1,0417	0,7366
CAN	9,7472	1,0009	3,1221

7.3 Test 3, fördröjning [CarMaker - Phase - CarMaker]

Det tredje testet mäter fördröjningen i systemet, från det att CarMaker skickar en signal till styrenheten, till att styrenheten skickar tillbaka det och att CarMaker har hunnit läsa in det. Utgående och inkommande värden plottas mot en tidsskala för att få ut fördröjningen mellan varje signal. Detta test genomförs på banan "Racetrack_Nurburgring" utan en extern förare. Simuleringen körs i 30 s, efter stopp sparas loggad data.

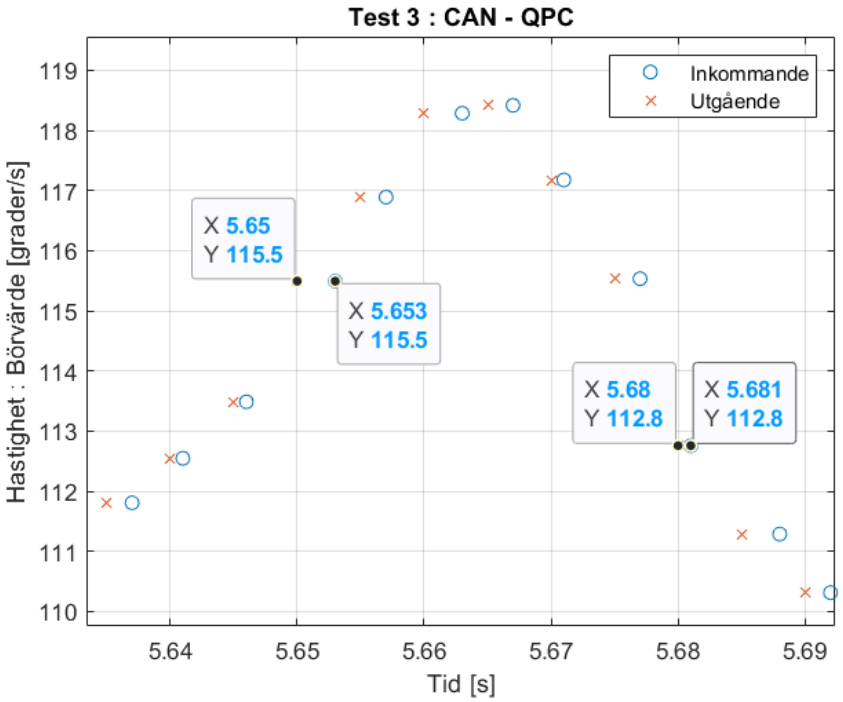
Test 3 genomförs på följande konfigurationer:
1. CarMaker HIL på National Instruments (facit/utgångsläge)
2. CarMaker Windows med stabiliserad cykeltid via CANoe över EtherCAT
3. CarMaker Windows med stabiliserad cykeltid över CAN (XL Driver Library)

Test 3 ger följande information:
1. Graf över både insignal och utsignal mot en tidsaxel.
2. Medelfördröjningen över varje skickad signal tills att den kommer tillbaka.
3. Beräkna standardavvikelse.

7.3.1 Test 3, Resultat

Test 3 visar skillnaden av medelfördröjningen mellan de olika konfigurationerna. Eftersom ingående och utgående värden för CAN är exakt samma i antal, går det att få ett exakt värde. Däremot är ingående och utgående värden för EtherCAT och HIL inte samma i antal, och därför görs stickprov.

Figur 18 visar utsignal(x) bredvid insignal(o) för test 3 med CAN. Medelskillnaden mellan insignal och utsignal enligt tabell 3 ger ett värde på 2,1, vilket även går att antyda i figuren.



Figur 18: Utsignal jämfört med insignal

För att få ett representativt värde av fördröjningen beräknas medelvärdet av fördröjningen mellan alla ut- och insignaler. Tabell 3 visar medelfördröjningen för att skicka och ta emot ett meddelande. Genom att dela fördröjningen med två antas det representera fördröjningen enkel väg.

Tabell 3: Jämförelse av fördröjning

	Medelfördröjning [ms]	Medelfördröjning Enkel väg [ms]	Standardavvikelse [ms]
HIL	7 (Stickprov)	3,5 (Stickprov)	-
EtherCAT	6 (Stickprov)	3 (Stickprov)	-
CAN	2,126	1,063	0,789

Då det inte finns möjlighet att mäta fördröjningen mellan styrenheten och servomotorn antas dess fördröjning vara samma som mellan CarMaker och styrenheten.
Den totala fördröjningstiden för hela systemet, CarMaker till servomotor(ratt) och tillbaka antas därför vara ~4 ms för CAN.

7.4 Test 4, PD-regulatorns styvhet

Det fjärde testet tar fram rimliga parametrar till PD-regulatorn i styrenheten. Det görs genom att beräkna styvheten i CarMakers simulerade styrsystem, benämns vidare CarMaker-systemet. CarMaker-systemet kan delas upp i tre delar vilket är rattstång, torsionsstav och styrväxel med hjulupphängning. Där rattstången och torsionsstaven har givna värden. CarMaker-system kan ses som seriekopplade fjädrar där totala styvheten för simuleringen fås genom att lägga ett simulerat moment på den simulerade ratten och läsa av dess vinkeländring. Styrenheten med den verkliga ratten kan sedan adderas på seriekopplingen av fjädrar, och bedömning av dess känsla utredas.

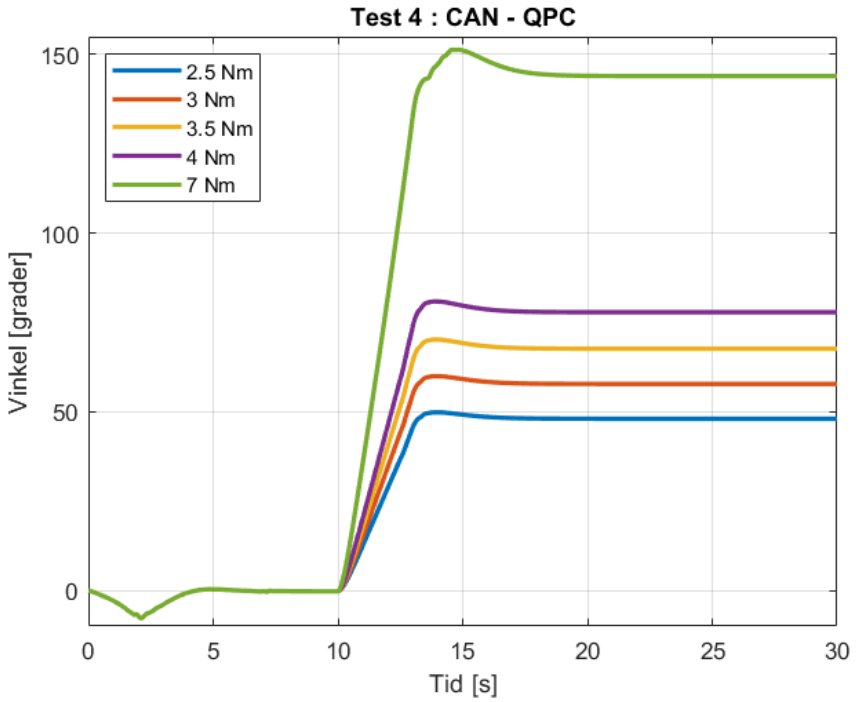
Testet genomförs på en tom parkeringsplats, utan en extern förare. Bilen kommer att köra rakt, tills en viss tidpunkt där momentet ökas till ett givet värde. Simuleringen loggas i 30 s.

Test 4 genomförs på följande konfiguration:
1. CarMaker Windows med stabiliserad cykeltid via CAN (XL Driver Library)

Test 4 ger följande information:
1. Tillräckligt med information för att beräkna K_{SmH}(styrväxel med hjulupphängningen).

7.4.1 Test 4, Resultat

Figur 19 visar den simulerade rattens vinkeländring beroende på vilket moment som läggs på. Alla test är satta till att nå sitt maxmoment efter 3 s. Förhållandet mellan maxmomentet och den slutliga vinkeländringen kan sedan användas till att beräkna K_{CM}.



Figur 19: 2.5-7 Nm med graf över vinklar

Tabell 4 visar sammanställt testresultat där givna värden är moment, styvhet rattstång och styvhet torsionstång. Avläst värde är vinkel. Beräknade värden är styvhet för CarMaker-systemet och styvhet för styrväxel med hjulupphängningen. Styvheten är för CarMaker-system benämns K_{CM}, rattstången K_{RS}, torsionstaven K_{TS} och styrväxel med hjulupphängning K_{SmH}.

Tabell 4: Beräknad styvhet vid olika moment och givna styvheter

Moment [Nm]	Vinkel [grader]	Styvhet K_{CM} [Nm/grad]	Styvhet K_{RS} [Nm/grad]	Styvhet K_{TS} [Nm/grad]	Styvhet K_{SmH} [Nm/grad]
2,5	48,06°	0,052	12	2	0,054
3,0	57,79°	0,052	12	2	0,054
3,5	67,66°	0,052	12	2	0,054
4,0	77,87°	0,051	12	2	0,053
7,0	144°	0,049	12	2	0,050

Styvheten för CarMaker-systemet, ratt till asfalt, beräknas genom att dividera pålagt moment med den vinkeländring som den simulerade ratten får. Genom att ta CarMaker-systemets styvhet minus de givna styvheterna i rattstången och torsionsstaven beräknas styvheten för styrväxel med hjulupphängning (1).

$$\frac{1}{K_{CM}} = \frac{1}{K_{SmH}} + \frac{1}{K_{RS}} + \frac{1}{K_{TS}} \tag{1}$$

Granskas resultatet av formel (1), noterade i tabell 4, framgår det att K_{SmH} är den avgörande faktorn i systemet. Vid en avrundning till två decimaler är K_{CM} lika med K_{SmH}.

$$\frac{1}{K_{CM}} \approx \frac{1}{K_{SmH}} \tag{2}$$

Då styvheten för rattstången och torsionstången har så låg påverkan på systemet, kan ett antagande göras att ifall styvheten för styrenheten ligger inom samma område, 2 - 12 Nm/grad, bör inte systemet påverkas av att styrenheten kopplas på. Genom ett räkneexempel där PD-regulatorns P-del(K_P) sätts till 4 och värden hämtas från tabell 4, beräknas den totala styvheten med formel (3) till 0,052 Nm/grad ($K_{CM\&P}$).

$$\frac{1}{K_{CM\&P}} = \frac{1}{K_{SmH}} + \frac{1}{K_{RS}} + \frac{1}{K_{TS}} + \frac{1}{K_P} \tag{3}$$

Beräkningsexempel av formel (3) vid 2,5 Nm:

$$\frac{1}{K_{CM\&P}} = \frac{1}{0{,}054} + \frac{1}{12} + \frac{1}{2} + \frac{1}{4} = \frac{1}{0{,}052} \tag{4}$$

Vid en avrundning till två decimaler är K_{CM} lika med $K_{CM\&P}$.

$$\frac{1}{K_{CM}} \approx \frac{1}{K_{CM\&P}} \tag{5}$$

Resultatet av ekvation (4) visar att inom givna värden av K_P ska den riktiga ratten representera den simulerade ratten.

7.5 Test 5, Sinuskörning

Det femte testet ger en överblick av systemet så att man enklare kan jämföra prestandaskillnaderna. Testet körs på en tom parkeringsplats med en mänsklig förare. Målet är att genom en metronom styra den verkliga ratten och eftersträva sinuskurvor i olika frekvenser. De frekvenserna som kommer att testas är 0 Hz, 0,5 Hz, 0,75 Hz, 1 Hz, 1,25 Hz och 1,5 Hz. EtherCAT testas inte av anledningen att det inte fungerade att logga vinkeln samtidigt som momentet skickas till CarMaker.

Test 5 genomföras på följande konfigurationer:

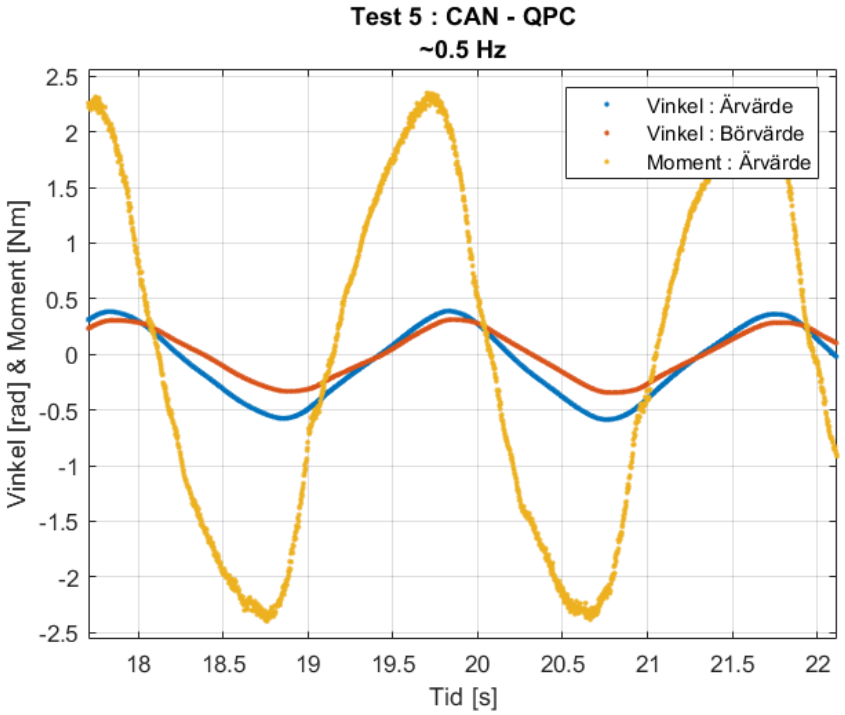
1. CarMaker HIL på National Instruments (facit/utgångsläge)
2. CarMaker Windows med stabiliserad cykeltid över CAN (XL Driver Library)

Test 5 ger oss följande information:

1. Graf över vinkelns börvärde och ärvärde samt moment för den verkliga ratten.
2. Graf över förstärkning och fasförskjutning.

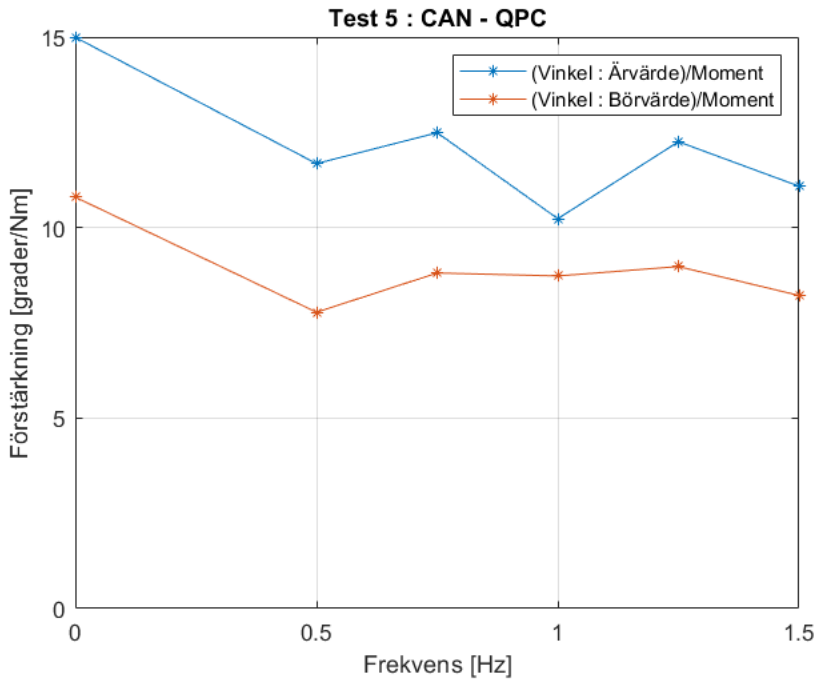
7.5.1 Test 5, Resultat

Figur 20 visar en förstorad bild av moment(Gul) jämfört med rattvinkelns ärvärde(Blå) och börvärde(Röd). Figuren visar testet kört i 0,5 Hz, eller 60 BPM. Det finns en amplitudskillnad mellan är- och börvärdet, samt en viss förskjutning vilket syns när de bryter Y=0.



Figur 20: Sinuskurva med 0,5 Hz med mänsklig förare

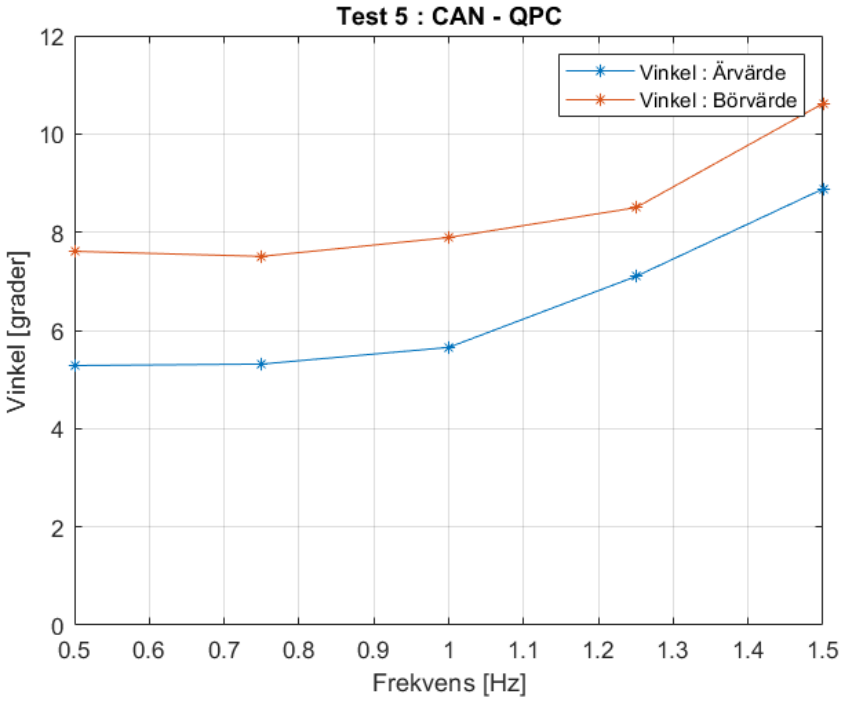
Förstärkningen och fasförskjutningen för de olika frekvenserna beräknas genom att ta ett snitt över 3 svängningar. Där Figur 21 visar förstärkningen mellan är- och börvärdet för systemet kört via CAN. Vid testet var K_P för styrenhetens regulator satt till 4.



Figur 21: Förstärkning CAN - QPC

Förstärkningen för CAN och HIL presenterade likvärdigt under testerna, se bilaga 1 för samtliga grafer.

Figur 22 visar fasförskjutningen mellan är- och börvärdet för systemet kört via CAN. För HIL ökar vinkelns börvärde men ärvärdet ändrats minimalt över frekvensspannet.



Figur 22: Fasförskjutning CAN - QPC

7.6 Test 6, Testkörning med förare(subjektivt test)

Det sjätte testet är ett subjektivt test. Det är för att ge en uppfattning av styrkänslan för de olika systemen, samt hur smidigt simuleringen uppfattas. Testet körs på banan "Racetrack_Nurburgring" i så lång tid som föraren vill, i en hastighet anpassad till banans svängar. Ingen mätdata kommer att loggas, men förarens uppfattning av simuleringen kommer antecknas.

Test 6 genomförs på följande konfigurationer:

1. CarMaker HIL på National Instruments (facit/utgångsläge)
2. CarMaker Windows med stabiliserad cykeltid via CANoe över EtherCAT
3. CarMaker Windows med stabiliserad cykeltid över CAN (XL Driver Library)

Frågor(skala 1 till 5):

1. Övergripande känslan.
2. Hur verkligt är upplevelsen jämfört med riktig bil?
3. Hur väl flyter simuleringen/körningen (fördröjning/lagg)?
4. Hur känns trögheten i ratten jämfört med en riktig bil?
5. Övrig kommentar:

7.6.1 Test 6, Resultat

Tabell 5 visar medelvärdet av fråga 1-4. Alla individuella svar hittas i bilaga 1, tillsammans med övriga kommentarer. Den gemensamma och övergripande känslan var att HIL-versionen var det som gav bäst upplevelse. Simuleringen uppfattades flyta på bättre och kännas lite mer verklighetstrogen. Men strax därefter hamnade både EtherCAT- och CAN-versionen som uppfattades nästintill lika bra. De kunde i vissa fall uppfattas lagga lite mer, men inte mycket mer.

Tabell 5: Medelvärde av fråga 1-4

	Fråga 1 Medelvärde	Fråga 2 Medelvärde	Fråga 3 Medelvärde	Fråga 4 Medelvärde
HIL	4,25	3,5	3,5	4,75
EtherCAT	3,75	3,25	3	3,5
CAN	4	3	3	3,5

8. Slutsats och diskussion

I detta kapitel kommenteras resultaten, frågeställningar från inledningen besvaras och vad som kunde gjorts bättre och framtida förbättringsmöjligheter beskrivs. Projektets kommenteras även kort ur ett hållbarhetsperspektiv.

8.1 Kommentarer kring uppsättning av system

Först och främst har användningen av Cockpit för att sköta kontakten med styrenheten fungerat väldigt bra. Det har funnits tydliga manualer som gått igenom hur programmet fungerar, samt vad alla funktioner gör. Något som däremot märktes var att klockfrekvensen på styrenhetens processor inte var tillräckligt hög för att det skulle vara helt optimalt att använda CAN som protokoll. Genom en ändrad inställning lyckades processorns frekvensen ändras från 80 MHz till 120 MHz. Det gjorde att det gick skicka en signal från CarMaker till styrenheten varje ~4 ms istället var ~9 ms. Tillverkaren gav inte heller tydliga svar om hur styrenheten kan konfigureras för att möjliggöra ett signalintervall på 1 ms. Samtliga tester fick därav genomföras med intervallet ~4 ms. I slutet av arbetet var det dock möjligt att skicka en signal varje 1 ms, utan att överbelasta styrenheten. Varför denna ändring skedde är okänd, och inget som hanns utredas eller genomföras tester med.

Att sätta upp systemen via CANoe för testning har fungerat bra. Det har varit ett väldigt smidigt program där tydlig hjälp och uppbackning givits av tillverkarna på Vector. Vid uppsättning av CAN stöttes det inte på några större problem som inte gick att få hjälp med, men ibland kunde det ta lite tid att sätta sig in i och förstå hur det behövdes konfigureras. Vid uppsättning av EtherCAT var det däremot större problem som inte var lika triviala. Ett problem var att det inte alltid gick att få kontakt med styrenheten när CANoe försökte ansluta till bussen. Varför det var så förblev okänt och det kändes mer som att singla slant. Ett annat problem vid uppsättning mellan EtherCAT och CANoe var PDO:er, som ska skickas och tas emot på bussen. Ibland när konfigurationen byttes från CAN till EtherCAT kunde styrenheten bland annat välja att skicka position i stället för moment. Varför det gjorde så lyckades inte listas ut, men vid radering och sedan återuppbyggnad av PDO konfigurationen fungerade det igen. Det blev inget stort problem, men var omständligt och tidskrävande.

Att sätta upp ett slutet system mellan testriggen och CarMaker har gått bra, men det har inte varit utan sina svårigheter. Från början var planen att skapa en egen applikation för EtherCAT, på liknande sätt som gjordes för CAN. Anledningen till att det var målet var möjligheten att hoppa över användningen av tredjehands programvara som mellanhand för uppkoppling. Men till slut insågs att det inte gick att lösas med den projektets hårdvara, därav togs beslut att gå vidare med projektet utan den lösningen.

8.2 Kommentarer av resultat

Efter test 1 och 2 märktes det tydligt att det krävdes en stabiliserad cykeltid för att CarMaker skulle fungera väl på Windows. Genomsnittet av tidsdifferensen var fortfarande väldigt bra utan stabiliserad cykeltid, men skillnaden mellan max- och minvärde var alldeles för stor. Det gjorde att programmet kördes hackigt, och många av

mätvärdena var inkorrekta. Efter test 1 och 2 gjordes avvägningen att en stabiliserad cykeltid krävs för de resterande testerna.

Resultatet av test 3 var väldigt förvånande. Av HIL-, EtherCAT- och CAN-versionen gav CAN bäst resultat och HIL gav sämst när det kommer till fördröjningstid. Det är svårt att säga exakt vad det beror på, men en stor anledning är förmodligen att vår CAN applikation inte kräver någon tredjehandsprogramvara som mellanhand. Informationen kan läsas av direkt i CarMaker. En annan anledning kan vara att styrenheten hanterar signaler från EtherCAT och CAN olika snabbt. Att EtherCAT via CANoe var snabbare än HIL var extremt förvånande. Båda använder sig av en tredjehandsprogramvara. Eventuellt kan det bero på att HIL-versionen inte lägger lika stort fokus på att logga data som att köra simuleringen.

Test 4 tog fram information av vad förstärkningen K_P borde vara i regulatorn i styrenheten. Det gav oss det teoretiska svaret att K_P borde vara oändligt stort. Detta är inte möjligt i praktiken därför systemet hade blivit extremt instabilt. CarMakers simulerade torsion- och rattstång har 2 och 12 Nm/grad som styvhet medan hjulupphängningen har 0,05 Nm/grad. Därför är det rimligt om regulatorn har ett värde mellan 2 och 12 så efterliknas den simulerade ratten. Det gjordes alltså inga beräkningar eller tester för att få optimala värdena för PD-regulatorn i styrenheten för att få ett så optimalt reglersystem som möjligt, utan värden sattes efter vad som uppfattades okej.

Test 5 visar att förstärkningens differens är relativt konstanta vid olika frekvenser. Vid 1 Hz hade CAN en mindre differensskillnad. Det gjordes endast ett test med sinsukörning och om det hade gjorts fler hade det gått sett om det händer något med förstärkningen vid 1 Hz eller om det bara är en tillfällighet. Test 5 är beroende av att en förare ska svänga i en exakt frekvens med exakt amplitud, vilket inte alltid är lätt att genomföra.

Test 6 gjordes främst för att få möjligheten att subjektivt yttra något om körupplevelsen. Testet är väldigt begränsat i omfattning och antal deltagare. Men det gav en antydan om hur systemen upplevs. Att den egengjorda CAN kändes så lik HIL är något som fascinerar oss, och vi undrar hur upplevelsen hade varit ifall ett blindtest hade gjorts.

8.3 Besvarande av frågeställning

- Hur kan jitter minimeras i simuleringsprogrammet CarMaker?

  Om CarMaker körs direkt på Windows märks det tydligt att tidsdifferensen mellan varje cykel inte är konsekvent. Windows klarar att köra snabbare än 1 ms per cykel, och gör därför det. För att minimera jitter i simuleringen krävs antingen en tydlig prioriteringsordning som HIL-versionen har, eller en stabiliserad cykeltid som projektet har använt.

- Hur är prestandaskillnaden mellan en RTD och en PC vid simulering?

  En RTD presterar mycket bättre än vad en PC har möjligheten att göra. Medan en RTD ligger exakt på 1 ms per cykel, ligger Windows med en cykeltid på ungefär

0,95-1,05 ms. Däremot verkar tidsfördröjningen kunna vara lägre på Windows jämfört med HIL. Slutsatsen är att en HIL-version är bättre, men att en Windows-version fortfarande kan vara användningsbar.

- Hur är prestandaskillnaden mellan CAN och EtherCAT vid simulering?

  Jämförelsen i denna rapport blir inte helt rättvis eftersom CAN och EtherCAT inte hade samma möjlighet vid testning. CAN med en egen applikation är bättre än EtherCAT med en tredjehands programvara som mellanhand. Detta bör undersökas vidare med en egen applikation för EtherCAT med ett kort byggt för 64-bit version av Windows.

- Hur påverkas styrkänslan av testriggens regulator?

  Styrenhetens regulator sitter som en fysisk ratt på CarMakers simulerade ratt. PD-regulatorns parametrar påverkar den totala styvheten och dämpningen, men jämfört med CarMakers hjulupphängning ska den påverkan vara väldigt liten. Det bör alltså inte påverka styrkänslan för mycket vid bra inställningar.

8.4 Utvecklingsområden

Det som hindrat oss har varit att vi inte haft korrekt EtherCAT-kort. FC1121 som vi använde oss av är byggt för 32-bit system, medan CarMaker använder sig av 64-bit. Det gick fortfarande att använda EtherCAT via CANoe med en simulerad master, men det gick inte att bygga vår egen applikation på det sättet vi gjorde med CAN-buss. På grund av halvledarbristen gick det heller inte att få tag i rätt EtherCAT kort under tiden som detta arbete har sträckt sig över, vilket ledde till att vi inte kunde gå hela vägen med EtherCAT. Men vi ser mycket positivt till att använda EtherCAT för liknande framtida projekt, om rätt kort används.

För att få CarMaker att köras stabilt krävdes det en stabiliserad cykeltid låst till 1ms. Det fungerade bra så länge Windows inte startar eller körde många andra processer samtidigt som simuleringen var igång. Windows kan vara väldigt oförutsägbart i vilka processer som hanteras i vilken ordning, det borde undersökas hur man kan sätta CarMaker högre prioriterat än andra processer. Bland annat går det ändra prioriteringar i aktivitetshanteraren, och det går förmodligen även tilldela en fast del av processorn till att köra just CarMaker och inget annat. Detta kan eventuellt vara till stor fördel om man endast kör simuleringen, och inte vill använda datorn till annat. Det var även ett problem med att Windows egna antivirusprogram startade oförutsägbart och påverkade prestandan negativt. Att använda ett tredjeparts antivirusprogram som går att styra kan vara en lösning till det problemet.

8.5 Hållbarhet

Bilindustrin står för en stor del av världens utsläpp, och det är just därför Volvo Cars jobbar för att minska sin klimatpåverkan. För varje test som går att köra med simulering i stället för en fysisk bil minskar utsläppen i längden. Självklart måste man ta hänsyn till att en simulator behöver material för att konstrueras och konstant drar el under

användning, men det är väldigt lite jämfört med utsläpp när man testar riktiga bilar. Så varje test som körs i en simulering är bättre för miljön samt för företagets kostnad.

Även ur ett säkerhetsperspektiv är simulering bättre vid testning än med riktiga bilar. De krafter som används vid simulering är mycket lägre, och det finns mindre skaderisk vid tillverkningsfel. De fel som skulle kunna uppstå vid en simulering finns även vid testning med riktig bil, men leder inte till fysiska krockar.

Referenser

[1]: K. G. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer science and engineering," in Proceedings of the IEEE, vol. 82, no. 1, pp. 6-24, Jan. 1994, doi: 10.1109/5.259423.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=259423

[2]: B. Thomas, Modern reglerteknik. 5. uppl., Stockholm, Sverige: Liber, 2016.

[3]: CAN in Automation(CiA), "Service data object (SDO)" 2021. [Online]. https://www.can-cia.org/can-knowledge/canopen/sdo-protocol/, Hämtad: 2022-01-18.

[4]: CAN in Automation(CiA), "Process data object (PDO)" 2021. [Online]. https://www.can-cia.org/can-knowledge/canopen/pdo-protocol/, Hämtad: 2022-01-18.

[5]: Little Endian, Big Endian [Begrepp för beskrivning av byteordning inom datakonstruktion]. Tillgänglig: https://sv.wikipedia.org/wiki/Endian Hämtad: 2021-12-16.

[6]: EtherCAT Technology Group, "EtherCAT - Technology Overview," 2021. [Online]. https://ethercat.org/en/technology.html, Hämtad: 2021-10-02.

[7]: FC1100 and FC1121 Application Note (EtherCAT Slave Card) Version 2.0, Verl, Tyskland: Beckhoff Automation, u.d. [Online]. Tillgänglig: https://download.beckhoff.com/download/document/io/infrastructure-components/fc11xxen.pdf, Hämtad: 2021-10-02.

[8]: CAN in Automation(CiA), "CANopen – The standardized embedded network," 2021. [Online]. https://www.can-cia.org/canopen/, Hämtad: 2021-10-02.

[9]: CANoe Product Information, Stuttgart, Tyskland: Vector, u.d. [Online]. Tillgänglig: https://assets.vector.com/cms/content/products/canoe/canoe/docs/Product%20Informations/CANoe_ProductInformation_EN.pdf , Hämtad: 2021-11-29

[10]: XL Driver Library Manual Version 20.30, Stuttgart, Tyskland: Vector, u.d. [Online]. Tillgänglig: https://assets.vector.com/cms/content/products/XL_Driver_Library/Docs/XL_Driver_Library_Manual_EN.pdf, Hämtad: 2021-10-02.

[11]: Ax Drives Series Configurable Motion Control Platform: Phase Motion Control, u.d. [Online]. Tillgänglig: http://www.phase-automation.fr/wp-content/uploads/2019/01/10460-0-D-M-ENG.AxM-II.RefMan.pdf, Hämtad: 2021-10-02.

[12]: Phase U3 Natural Cooling, Genua, Italien: Phase Motion Control, u.d. [Online]. Tillgänglig: https://www.phase.eu/en/products/servo-motors/ac-servo-motors/natural-cooling/, Hämtad: 2021-12-16

[13]: Cockpit 3 User Manual: Phase Motion Control, u.d. [Online]. Tillgänglig: http://www.phase-automation.com/uploads/Documentation/AxX/48000000001-ENG-UserMan.pdf , Hämtad: 2021-11-29.

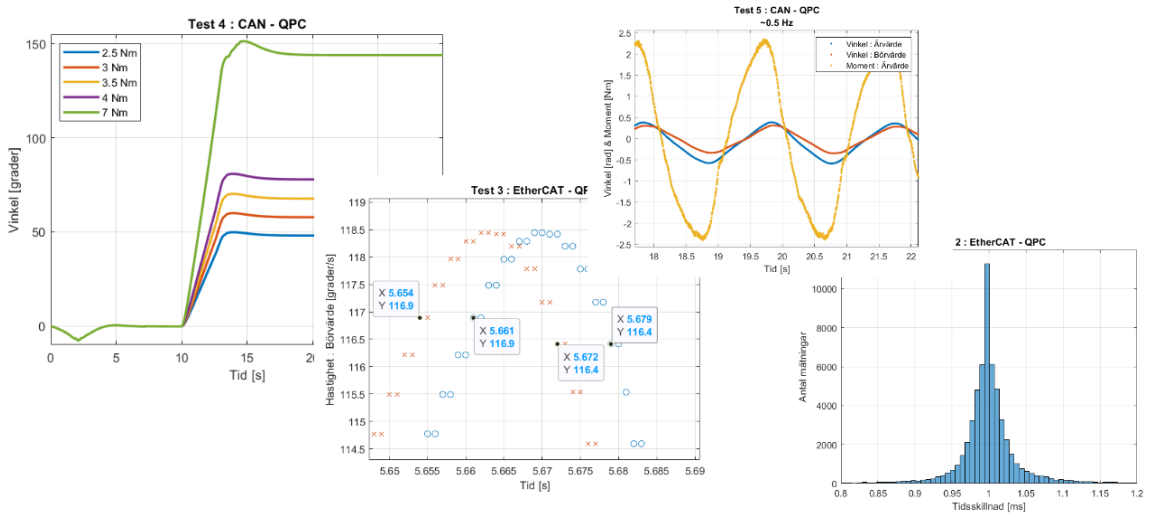
[14]: Microsoft Corporation, "Acquiring high-resolution time stamps," 2021. [Online]. Tillgänglig: https://docs.microsoft.com/en-us/windows/win32/sysinfo/acquiring-high-resolution-time-stamps, Hämtad: 2021-11-11.

[15]: CarMaker Programmer's Guide Version 10.1: IPG Automotive Group, u.d. [Manual inkluderad i CarMaker applikation Version 10.1] Hämtad: 2021-10-02.

[16]: Vector, "CANoe," 2021. [Online]. Tillgänglig: https://www.vector.com/se/en-se/products/products-a-z/software/canoe/, Hämtad: 2021-11-29.

[17]: CarMaker Reference Manual Version 10.1: IPG Automotive Group, u.d. [Manual inkluderad i CarMaker applikation Version 10.1] Hämtad: 2021-10-02.

Bilaga 1 - Grafer och data



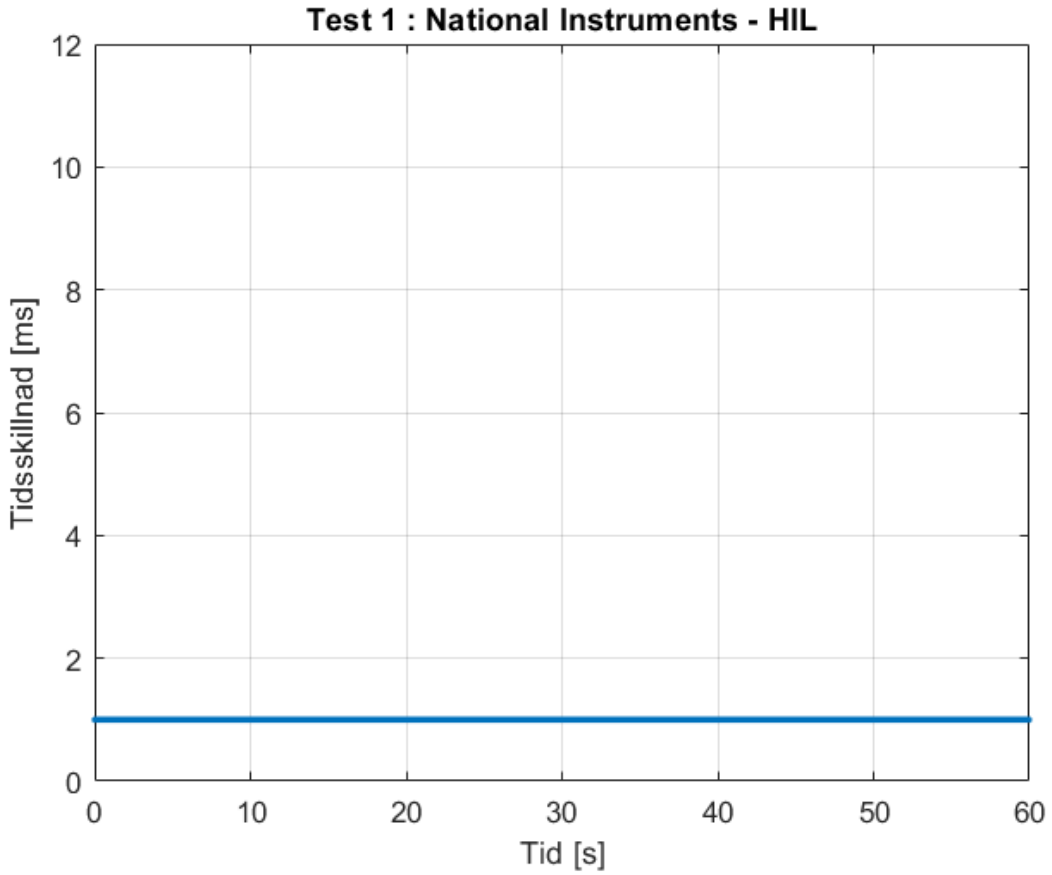
Innehållsförteckning

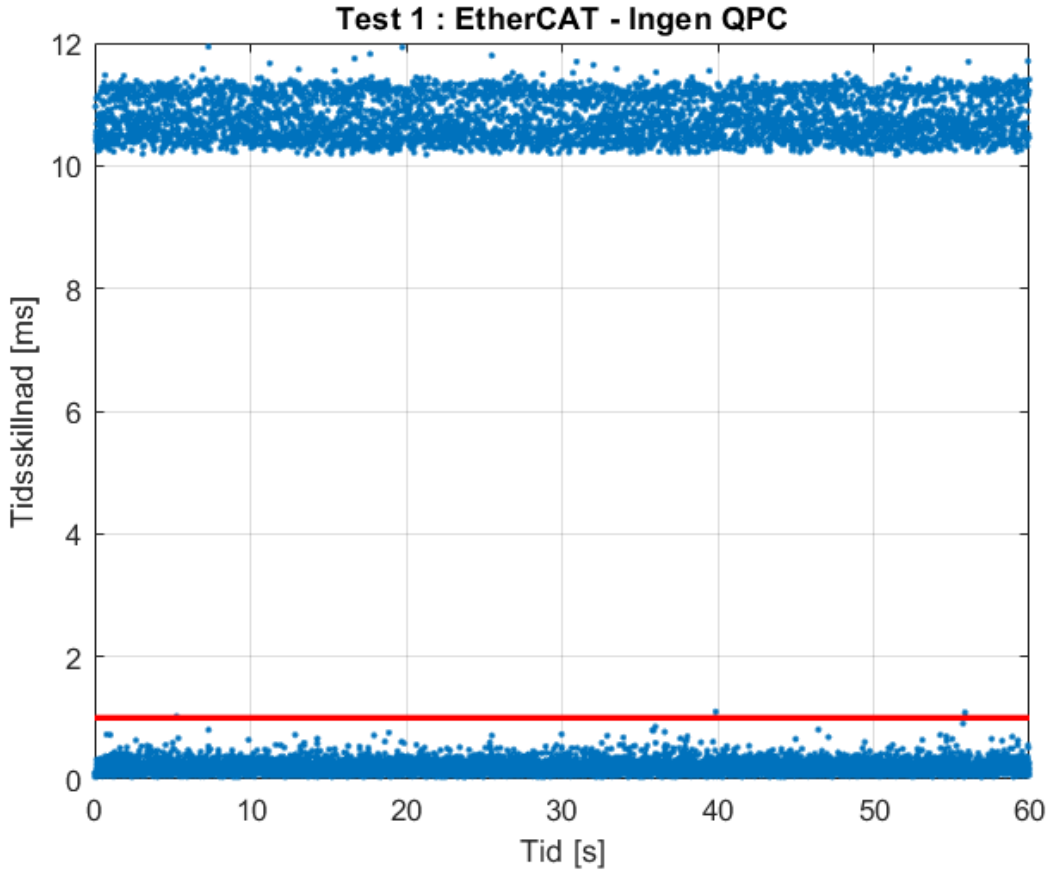
Test 1 - Grafer och data

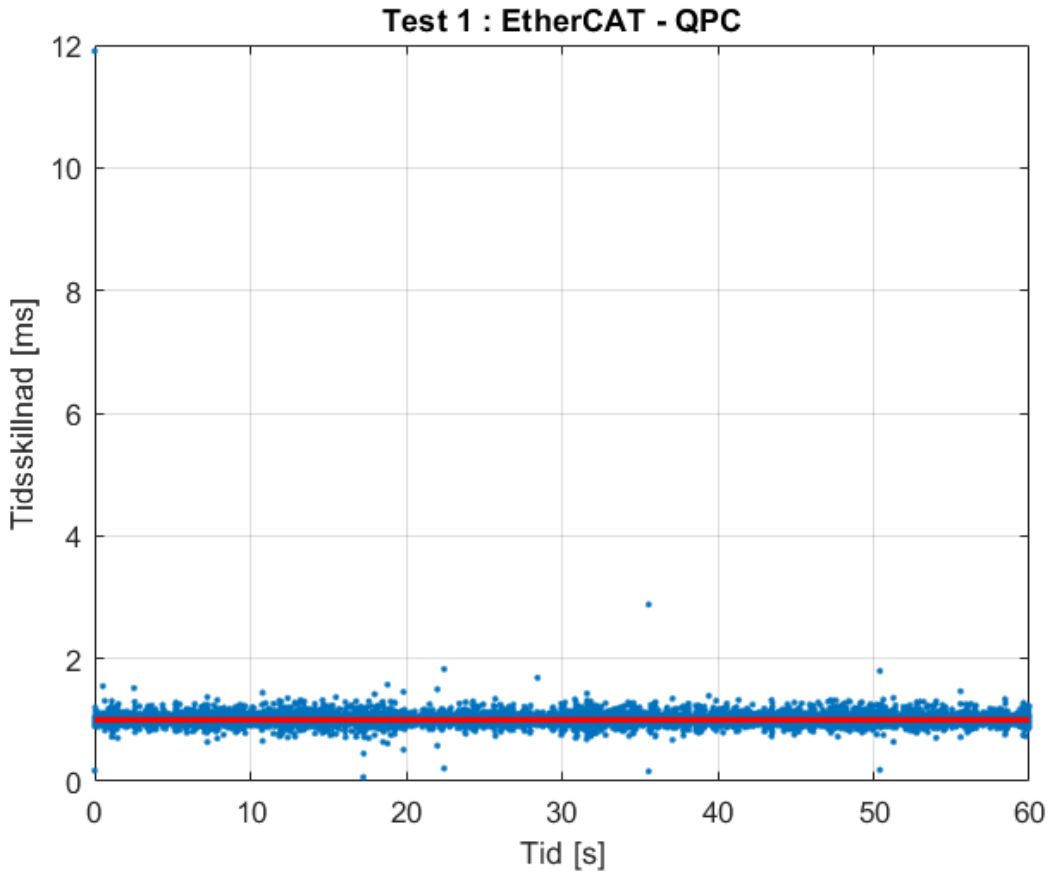
Tidsdifferens mellan varje cykel applikationen CarMaker gör.
MATLAB: plot(diff(x))

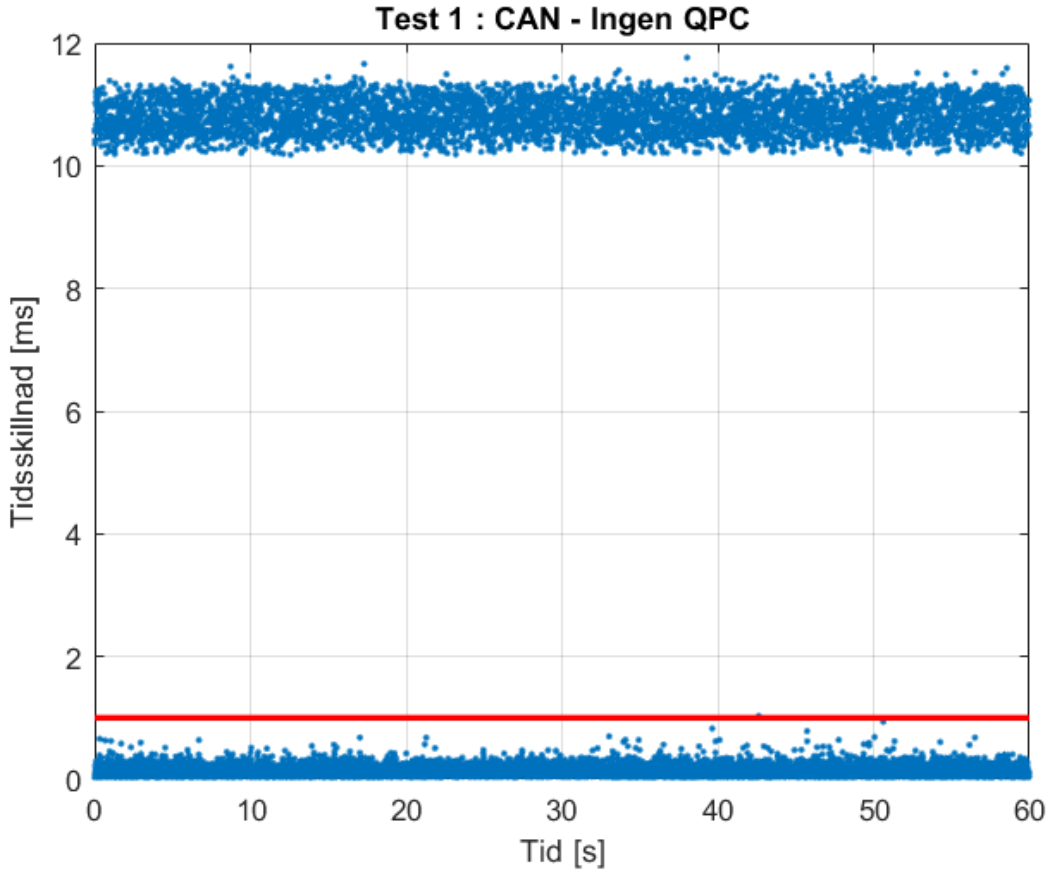
Tabell 1: Jämförelse av cykeltid

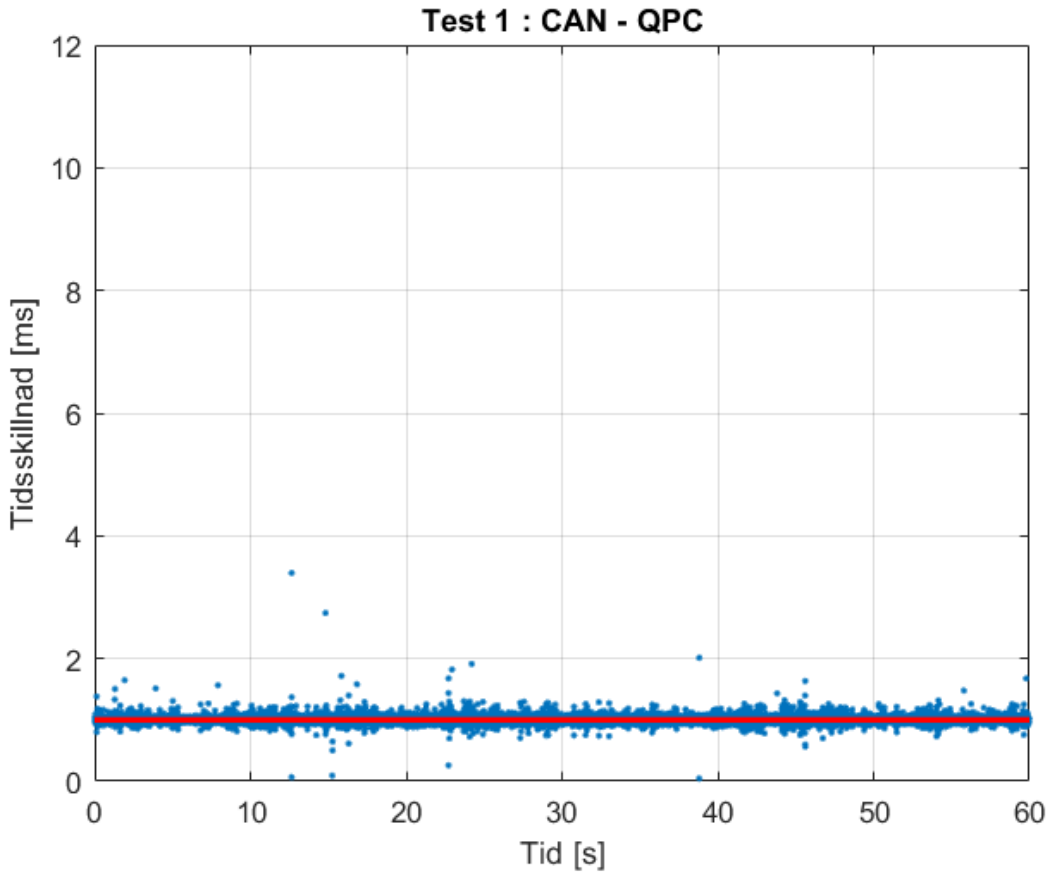
	Medelspridning [ms]	Medeltid [ms]	Standardavvikelse [ms]
HIL	$8{,}18 \cdot 10^{-24}$	1	$2{,}998 \cdot 10^{-12}$
EtherCAT - QPC	0,0579	1,002	0,2406
EtherCAT	8,4929	0,9992	2,9143
CAN - QPC	0,0463	1,0018	0,2152
CAN	8,8197	0,9999	2,9698



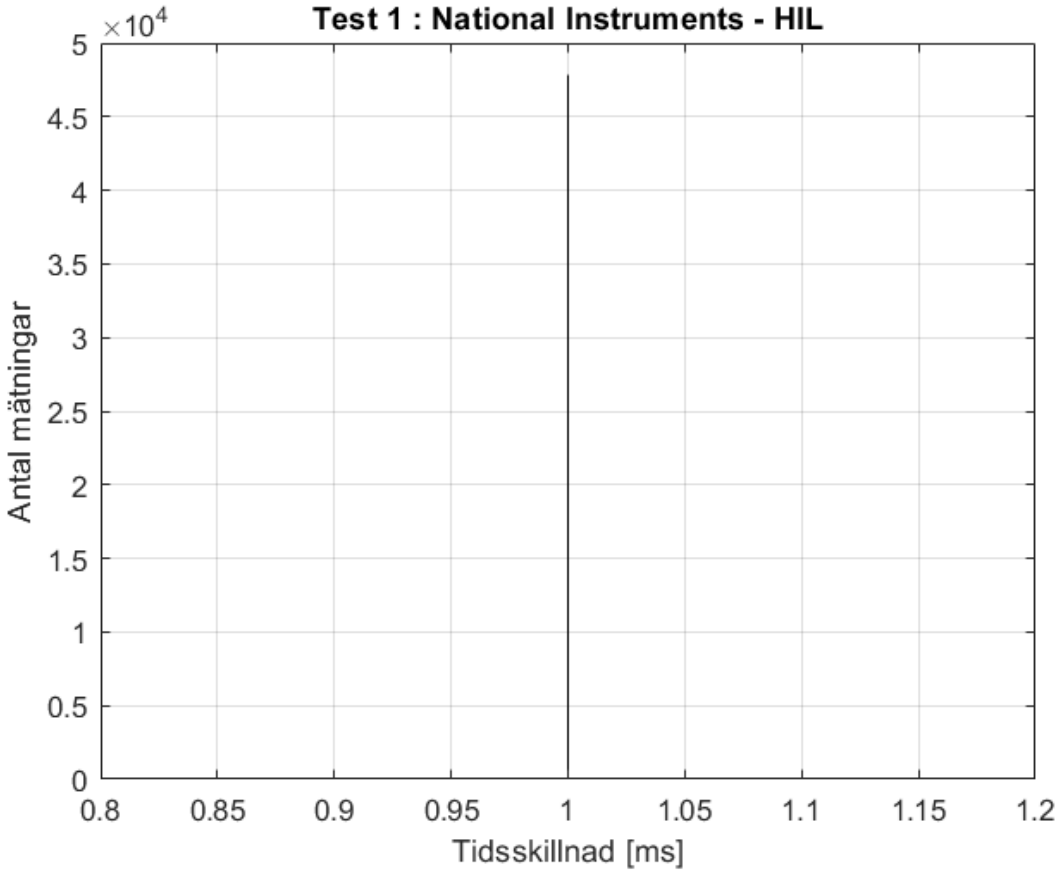

Test 1 : EtherCAT - Ingen QPC
Tidsskillnad [ms]
Tid [s]
0
2
4
6
8
10
12
0
10
20
30
40
50
60

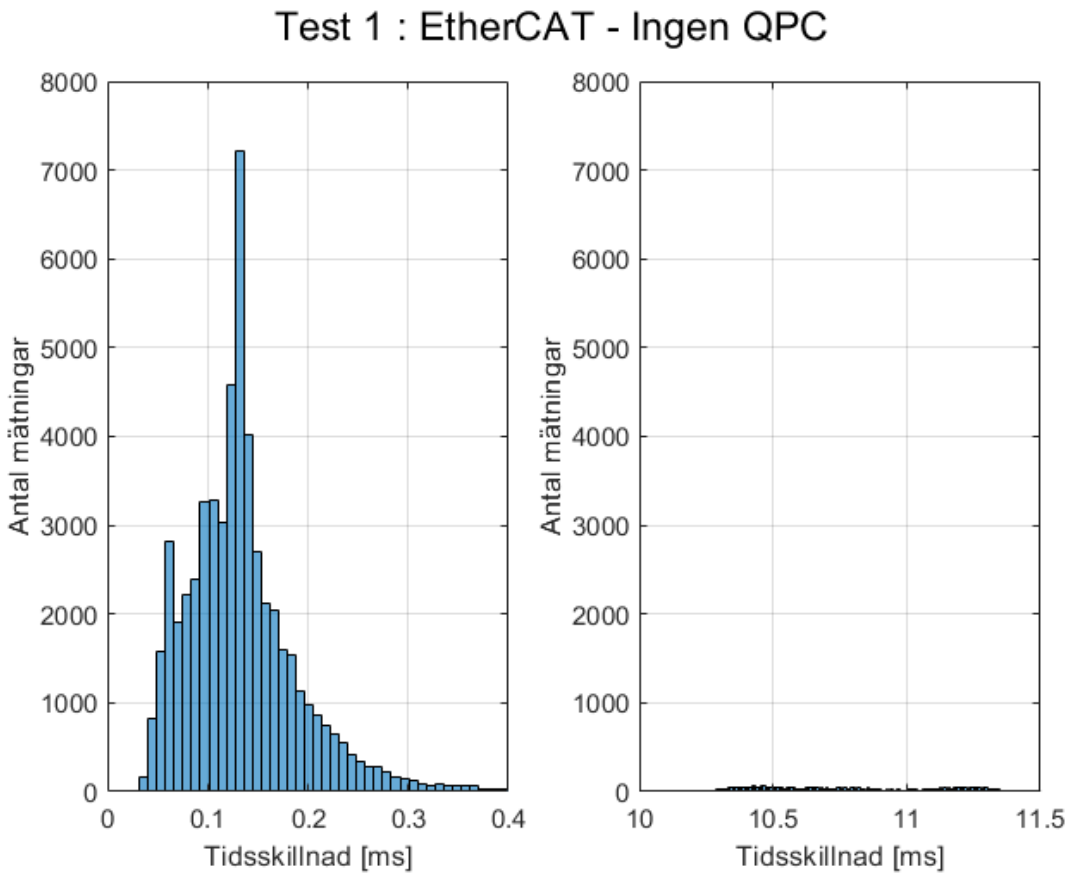

Test 1 : EtherCAT - QPC
Tidsskillnad [ms]
Tid [s]
0
2
4
6
8
10
12
0
10
20
30
40
50
60

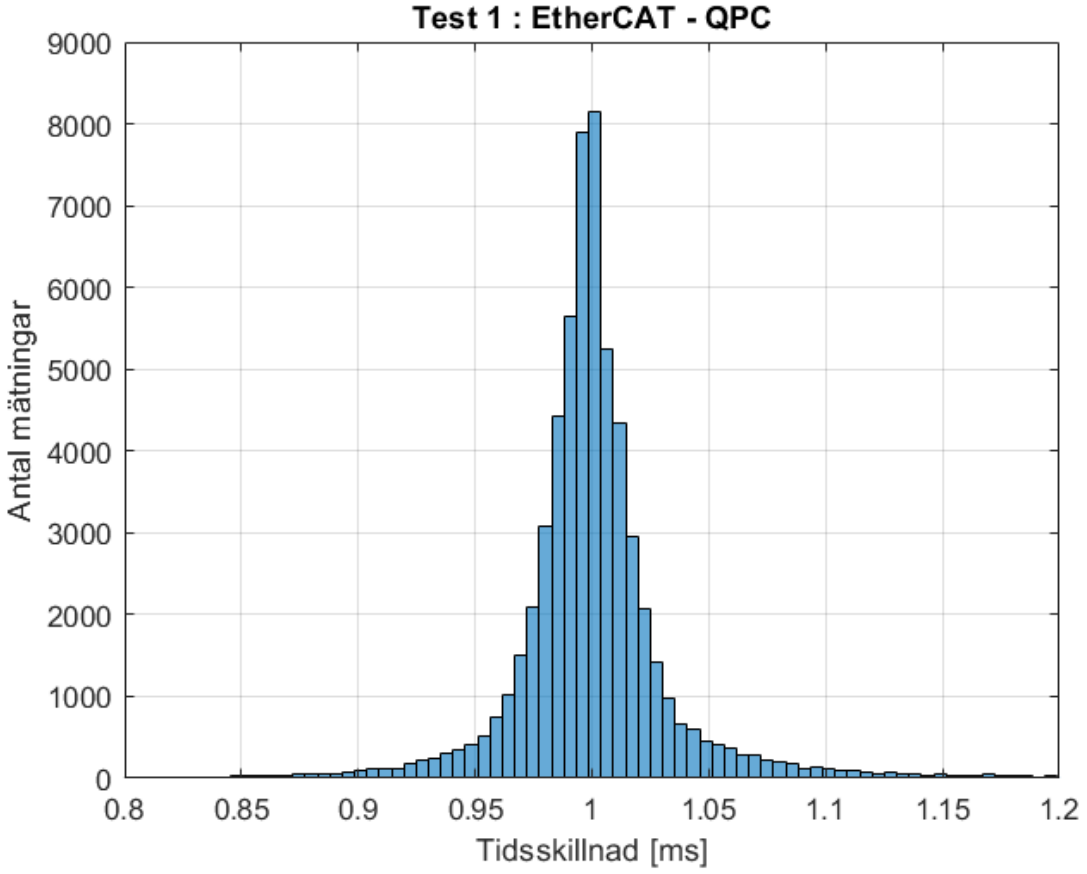

Test 1 : CAN - Ingen QPC
Tidsskillnad [ms]
Tid [s]
0
2
4
6
8
10
12
0
10
20
30
40
50
60

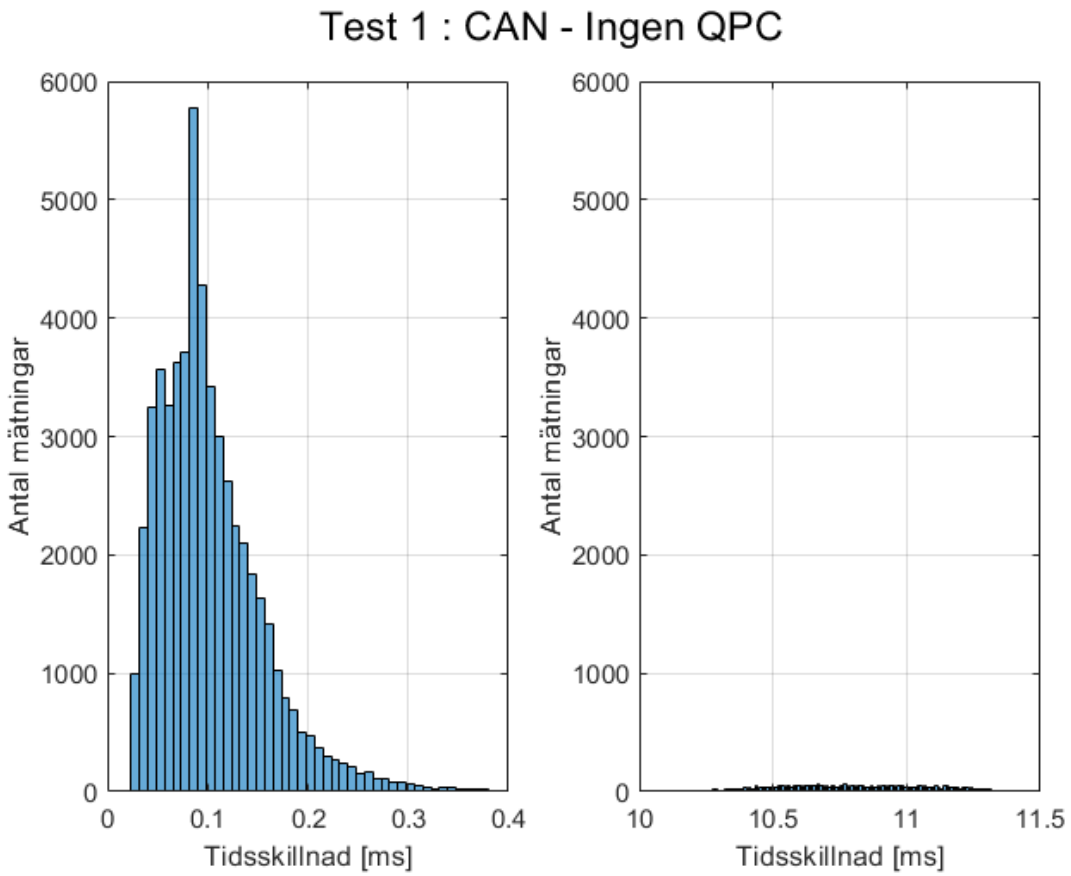

Test 1 : CAN - QPC
Tidsskillnad [ms]
Tid [s]
0
2
4
6
8
10
12
0
10
20
30
40
50
60

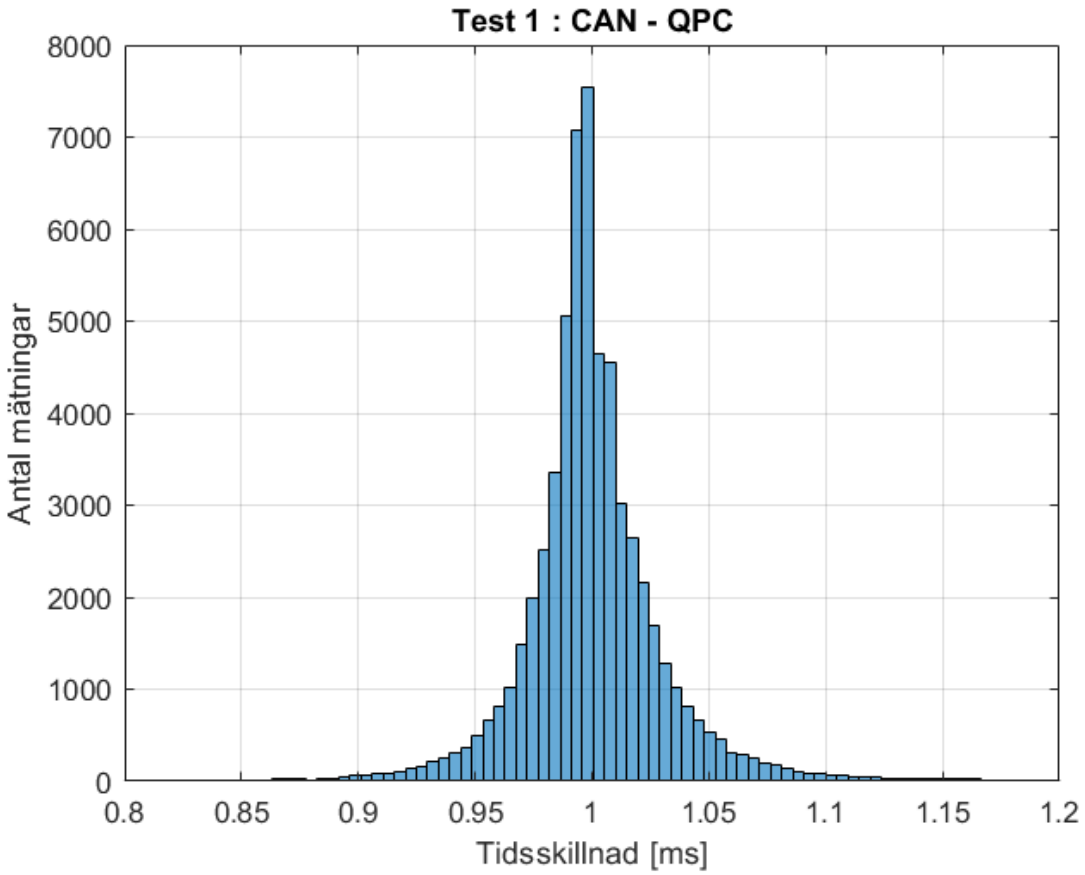
Histogram över tidsdifferens mellan varje cykel applikationen CarMaker gör.
MATLAB: histogram(diff(x))




Test 1 : EtherCAT - Ingen QPC
Antal mätningar
Tidsskillnad [ms]
0
1000
2000
3000
4000
5000
6000
7000
8000
0.1
0.2
0.3
0.4
10
10.5
11
11.5


Test 1 : EtherCAT - QPC
Antal mätningar
Tidsskillnad [ms]
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
0.8
0.85
0.9
0.95
1
1.05
1.1
1.15
1.2


Test 1 : CAN - Ingen QPC
Antal mätningar
Tidsskillnad [ms]
Antal mätningar
Tidsskillnad [ms]

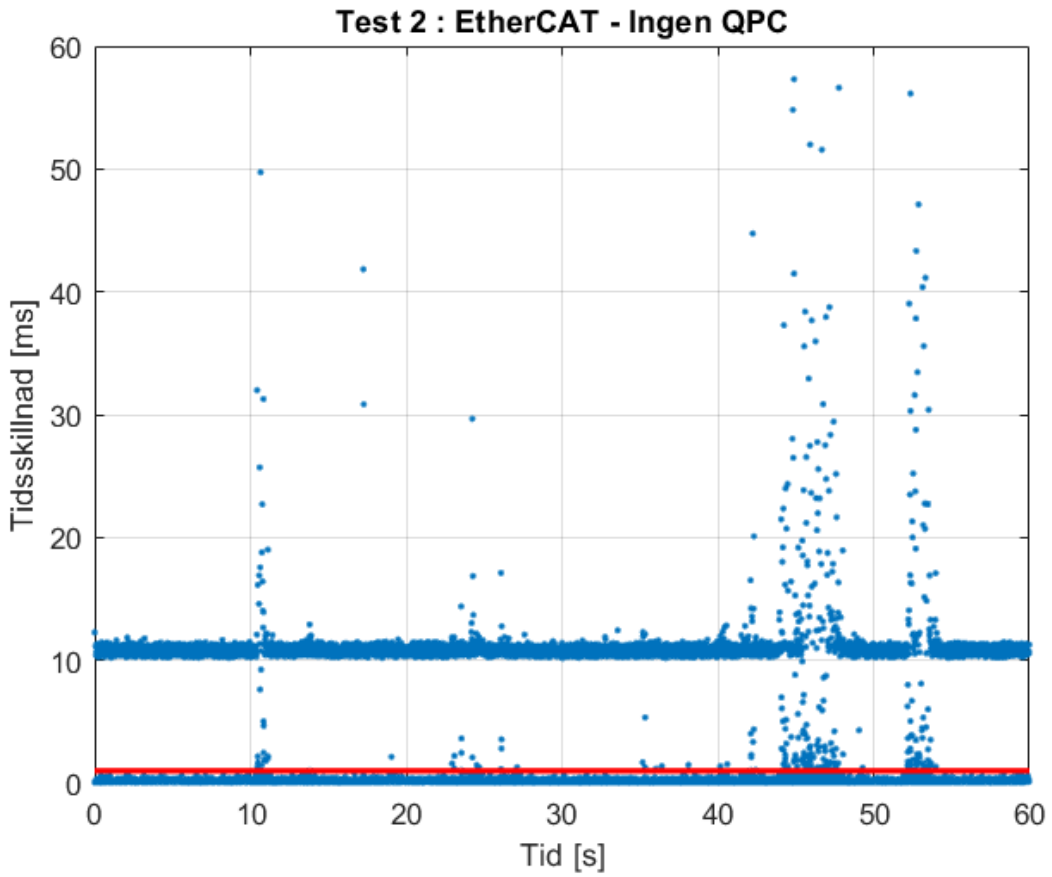

Test 1 : CAN - QPC
Antal mätningar
Tidsskillnad [ms]

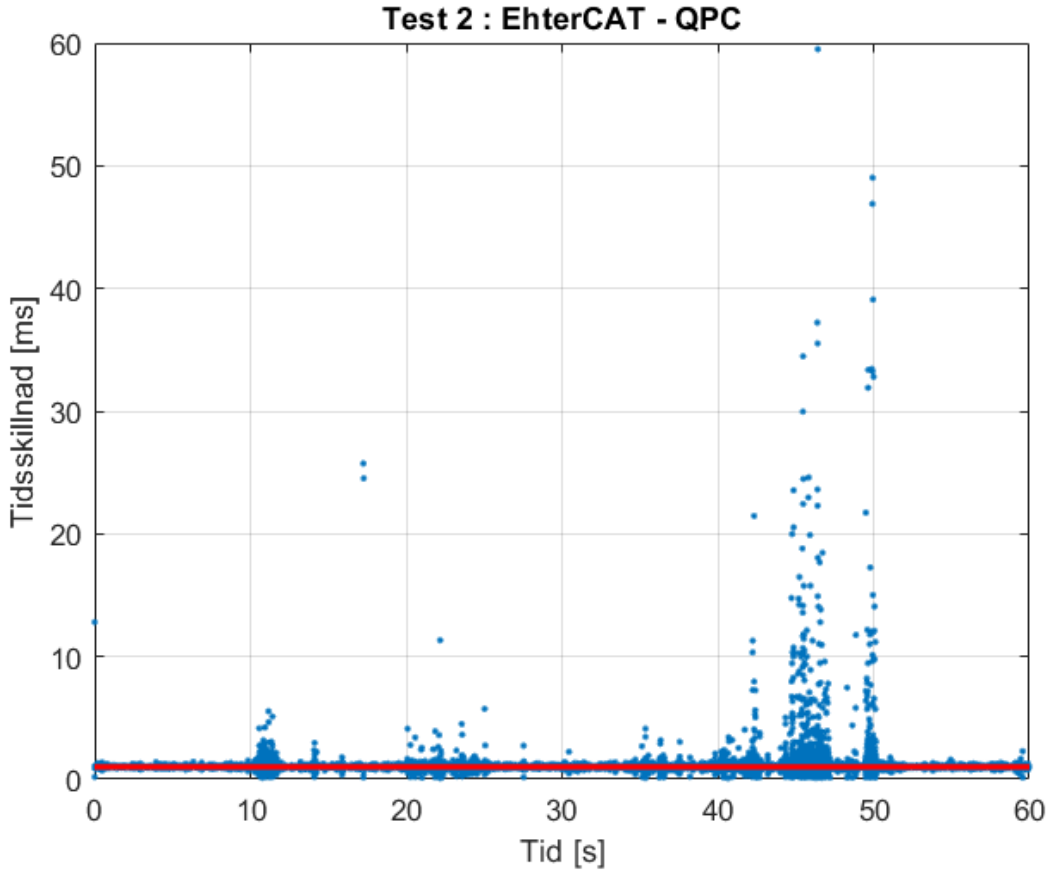
Test 2 - Grafer och data

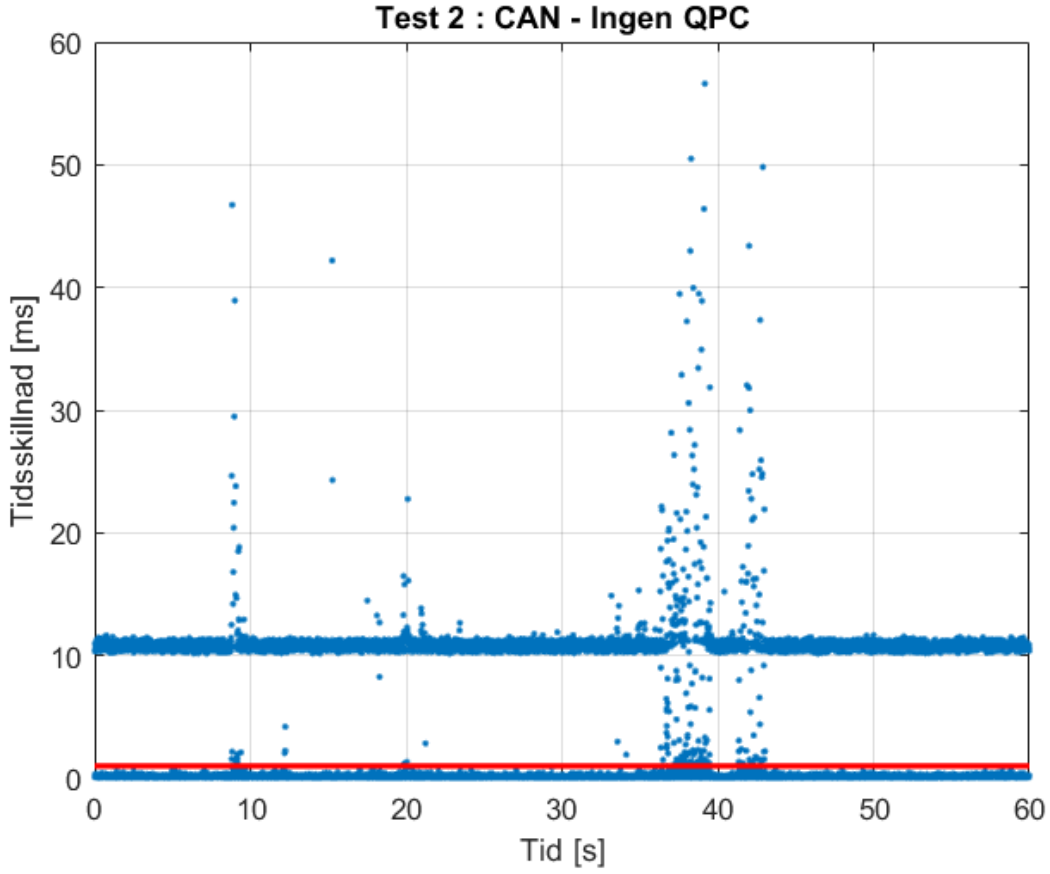
Tidsdifferens mellan varje cykel applikationen CarMaker gör under stress av annan applikation.
MATLAB: plot(diff(x))

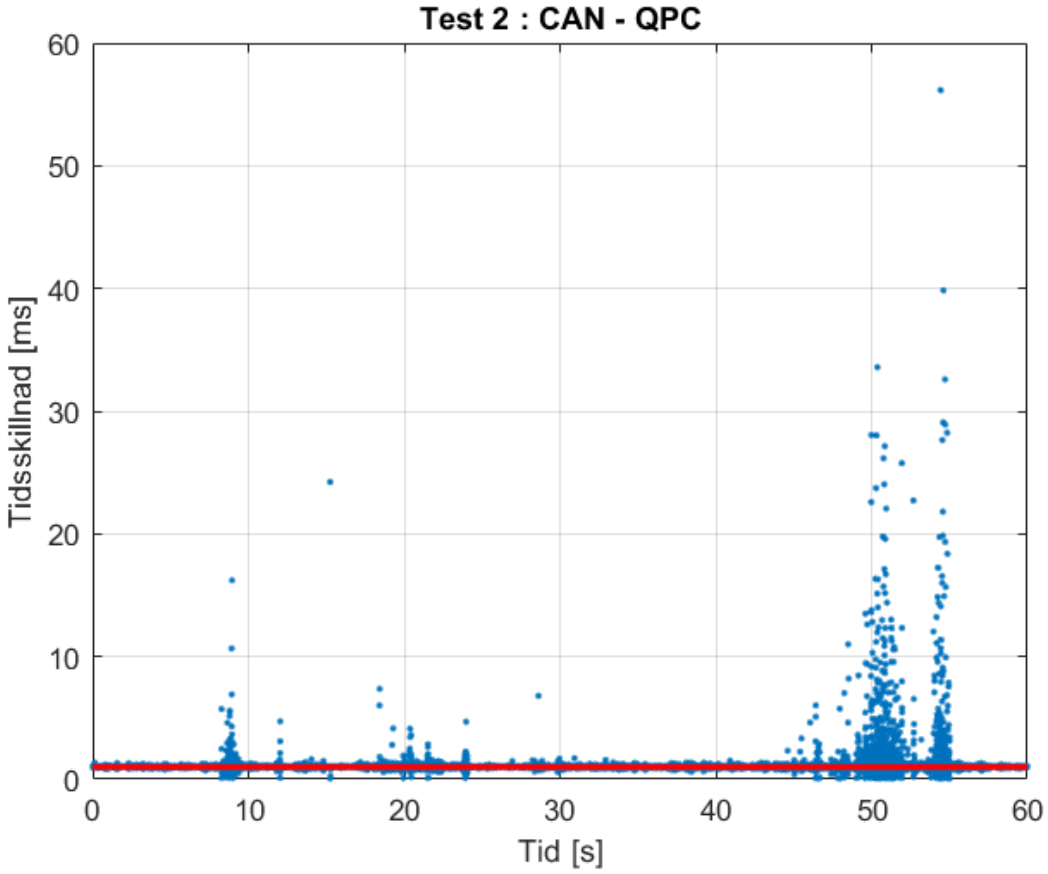
Tabell 2: Jämförelse av cykeltid vid överbelastning

	Medelspridning [ms]	Medeltid [ms]	Standardavvikelse [ms]
EtherCAT - QPC	0,6903	1,0396	0,8308
EtherCAT	10,6414	1,0074	3,2621
CAN - QPC	0,5426	1,0417	0,7366
CAN	9,7472	1,0009	3,1221

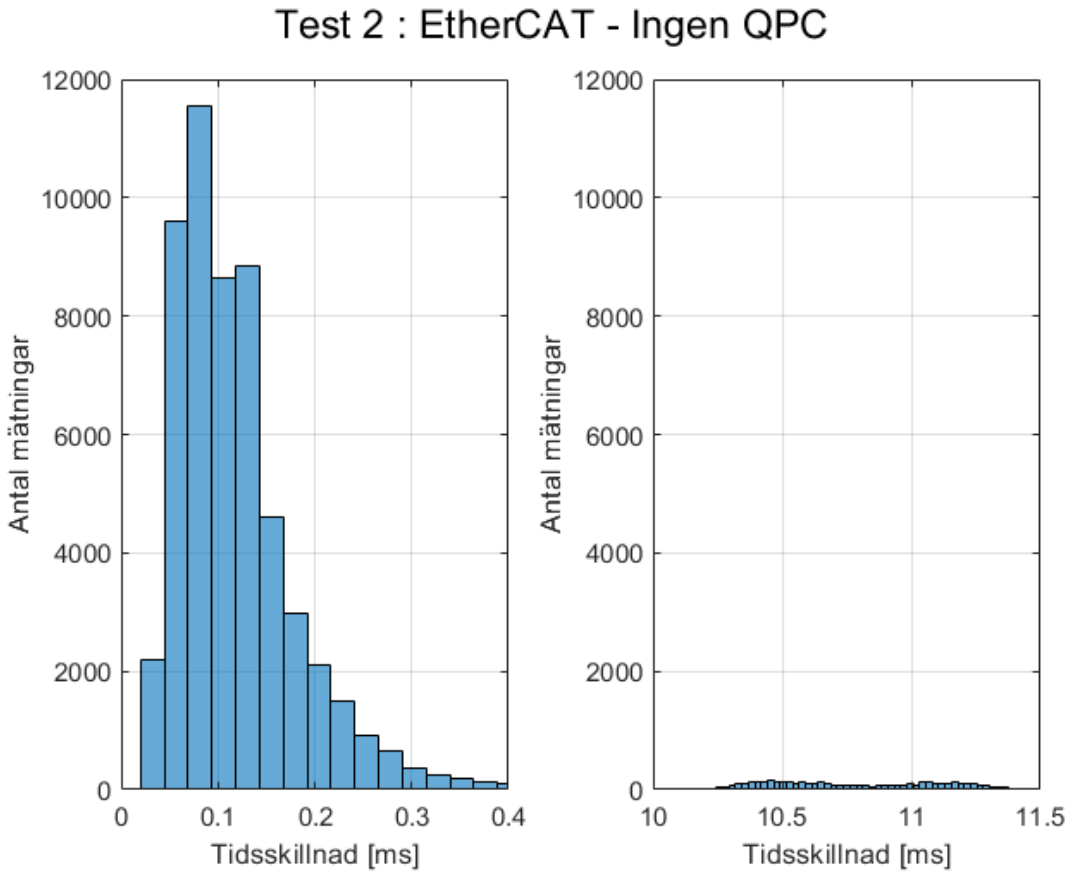


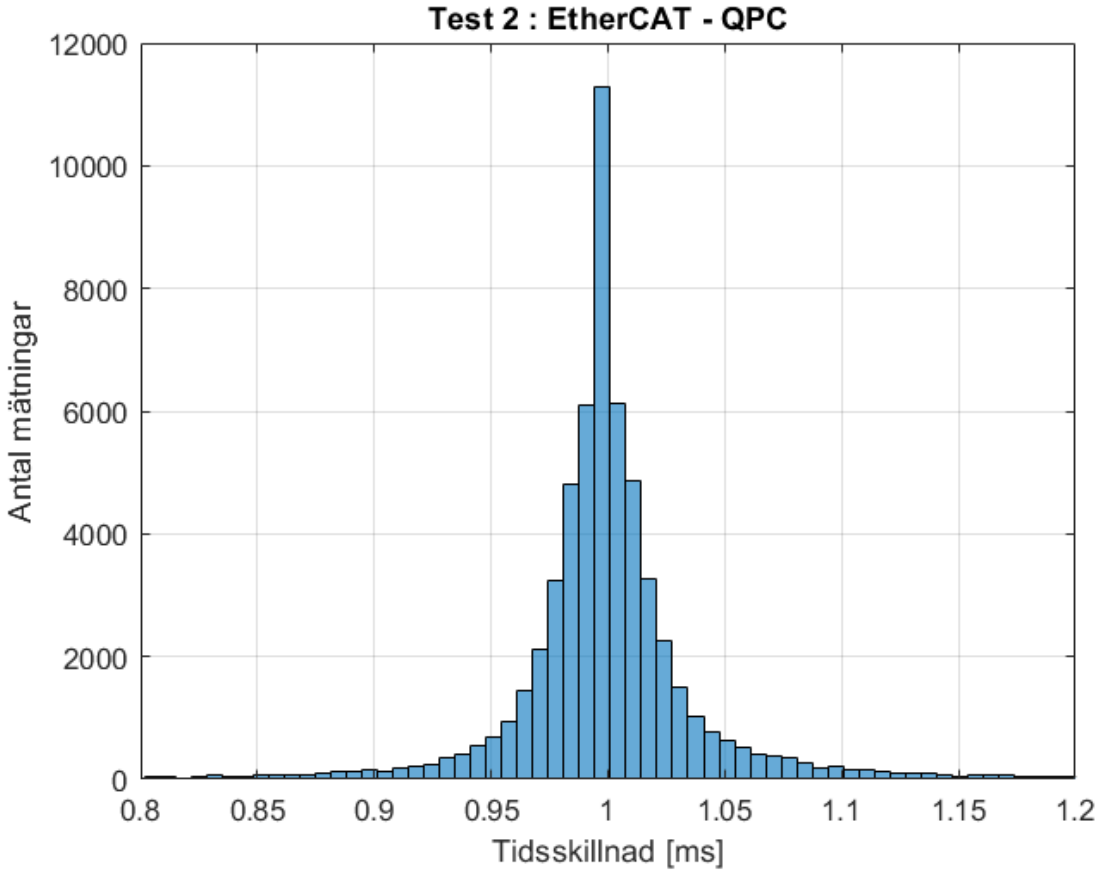

Test 2 : EhterCAT - QPC
Tidsskillnad [ms]
Tid [s]
0
10
20
30
40
50
60

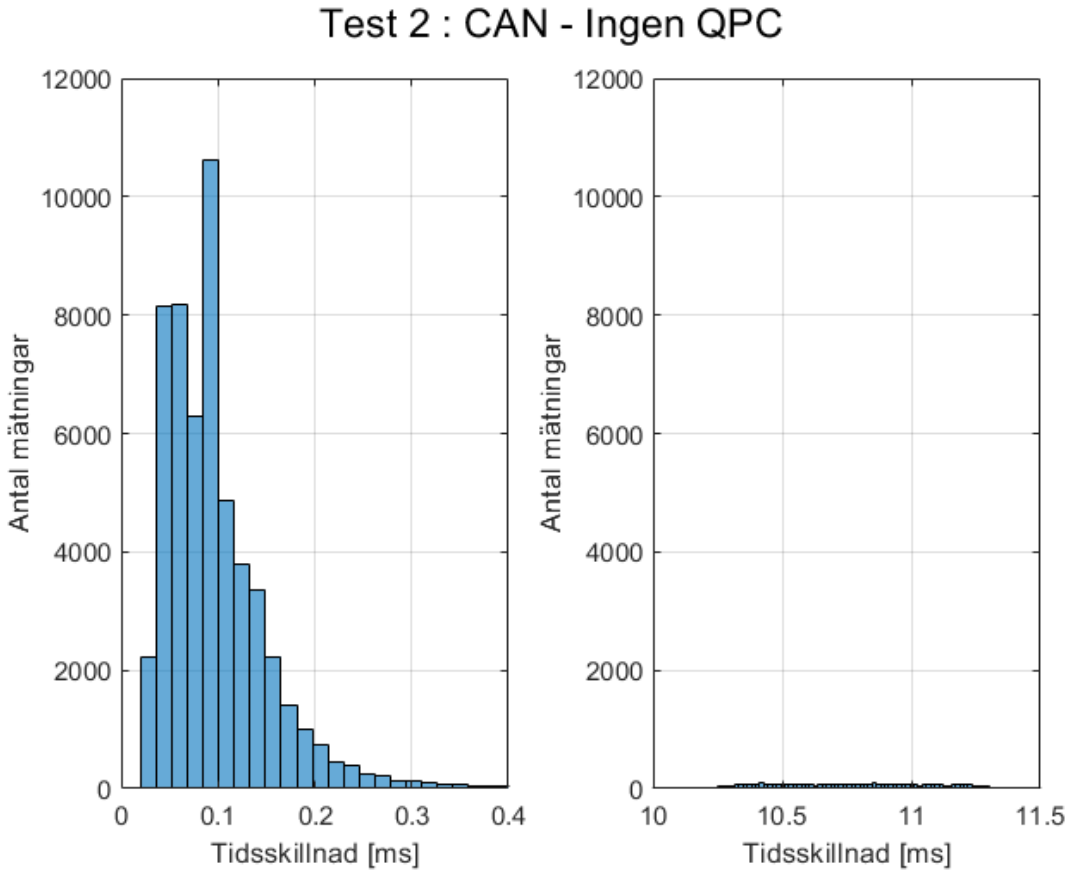

Test 2 : CAN - Ingen QPC
Tidsskillnad [ms]
Tid [s]
0
10
20
30
40
50
60

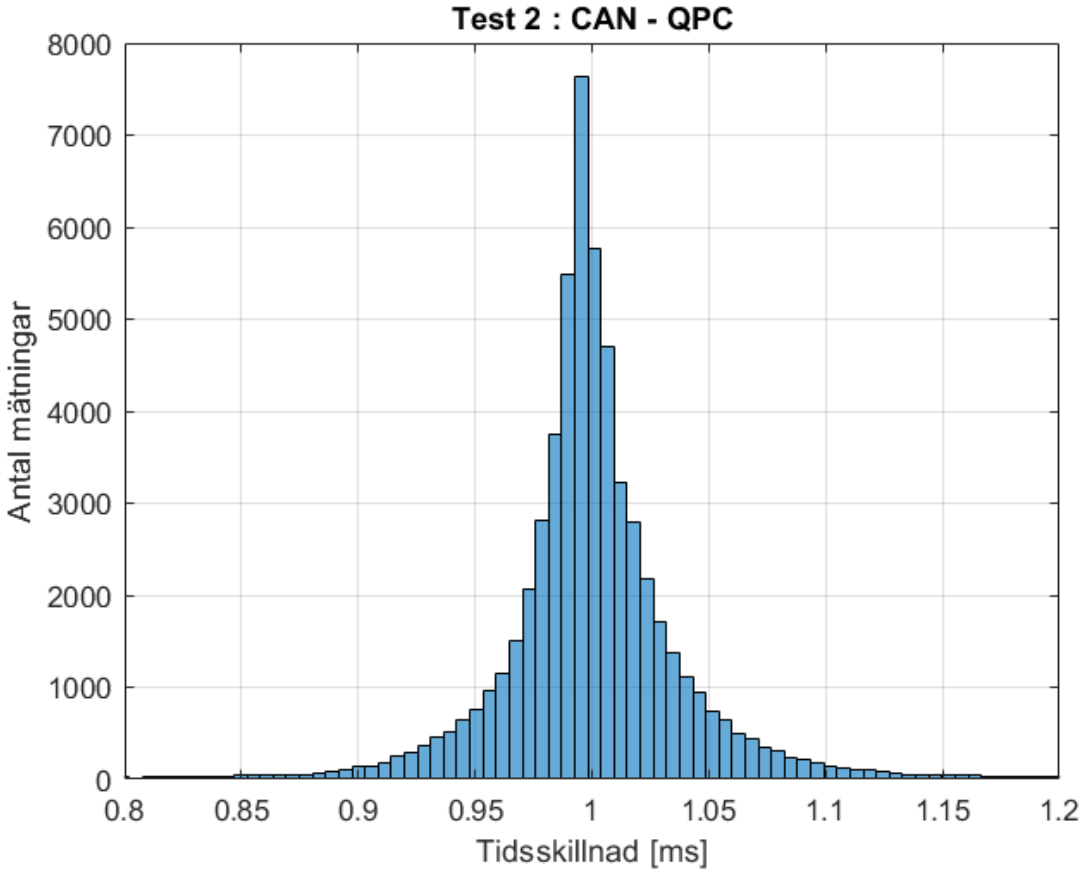

Test 2 : CAN - QPC
Tidsskillnad [ms]
Tid [s]
0
10
20
30
40
50
60

Histogram över tidsdifferens mellan varje cykel applikationen CarMaker gör under stress av annan applikation.
MATLAB: histogram(diff(x))




Test 2 : EtherCAT - QPC
Antal mätningar
Tidsskillnad [ms]
0
2000
4000
6000
8000
10000
12000
0.8
0.85
0.9
0.95
1
1.05
1.1
1.15
1.2


Test 2 : CAN - Ingen QPC
Antal mätningar
Tidsskillnad [ms]
0
2000
4000
6000
8000
10000
12000
0
0.1
0.2
0.3
0.4
10
10.5
11
11.5

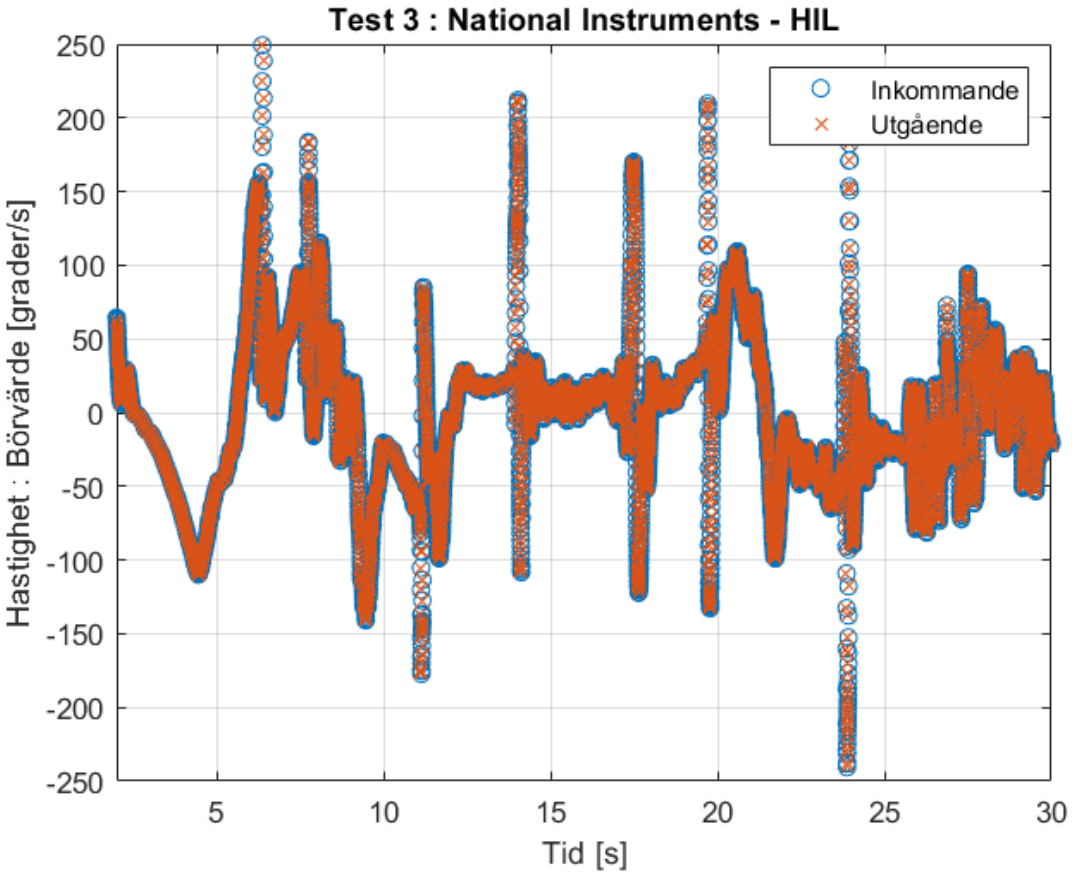

Test 2 : CAN - QPC
Antal mätningar
Tidsskillnad [ms]
8000
7000
6000
5000
4000
3000
2000
1000
0
0.8
0.85
0.9
0.95
1
1.05
1.1
1.15
1.2

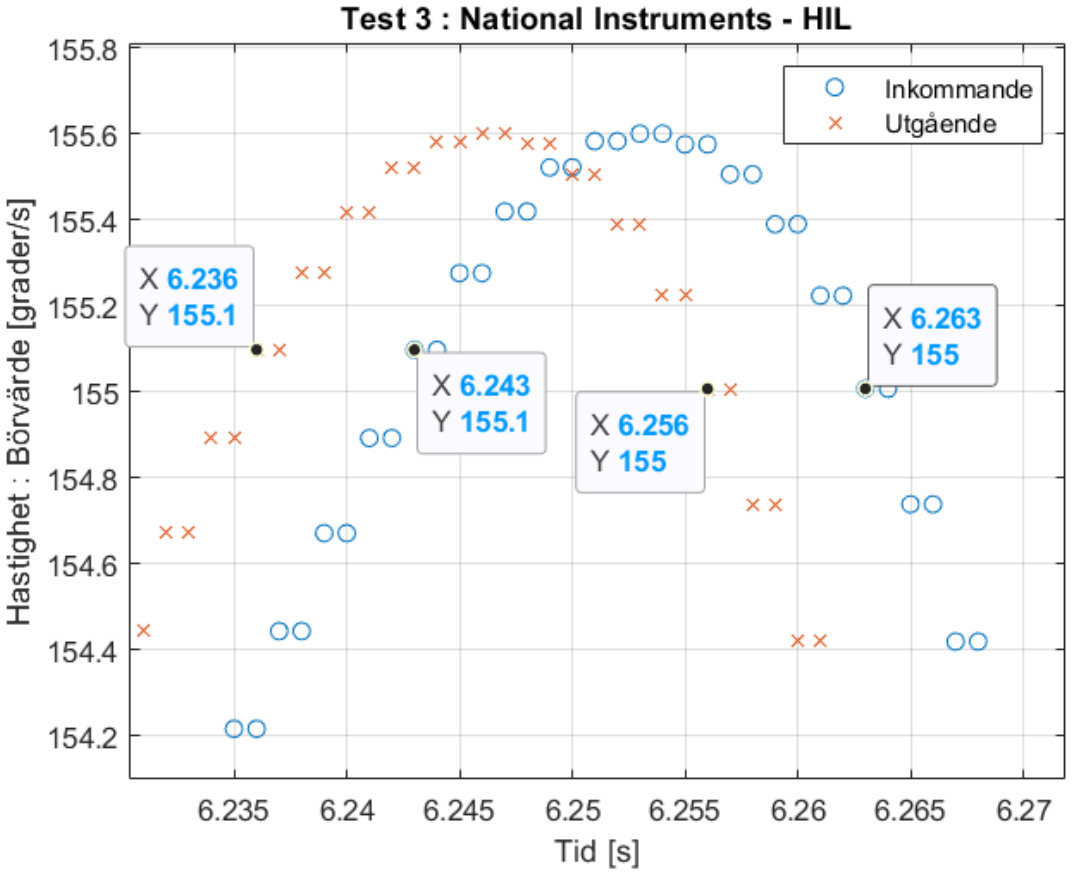
Test 3 - Grafer och data

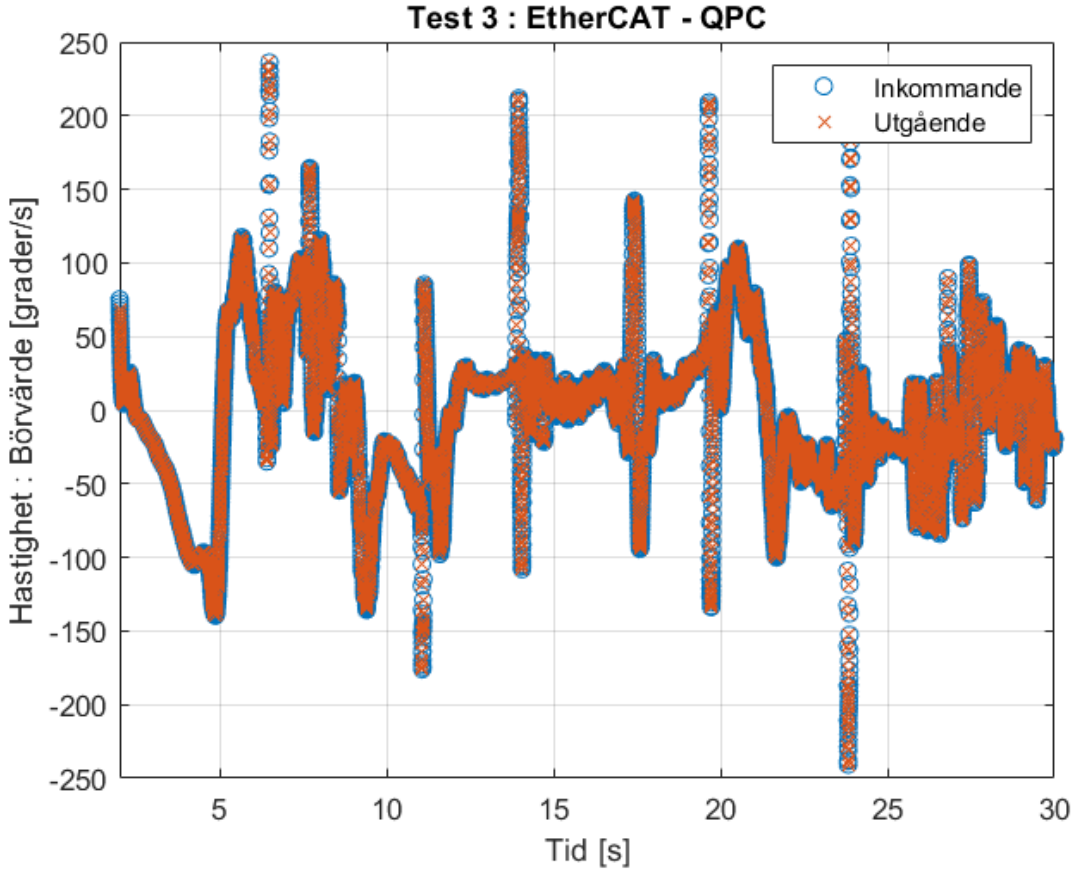
Tidsdifferens mellan utgående värde och inkommande, CarMaker till Styrservo till CarMaker.
Matlab: plot(tidut, yut, tidin, yin)

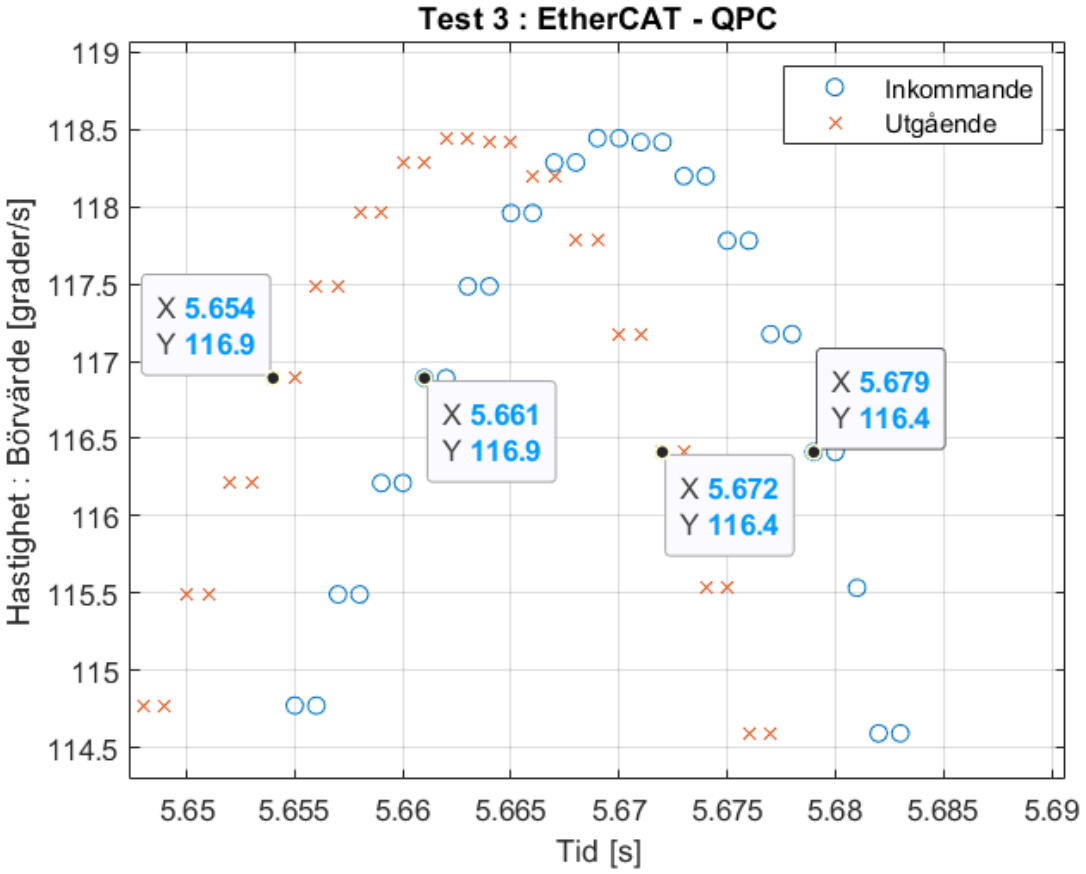
Tabell 3: Jämförelse av fördröjning

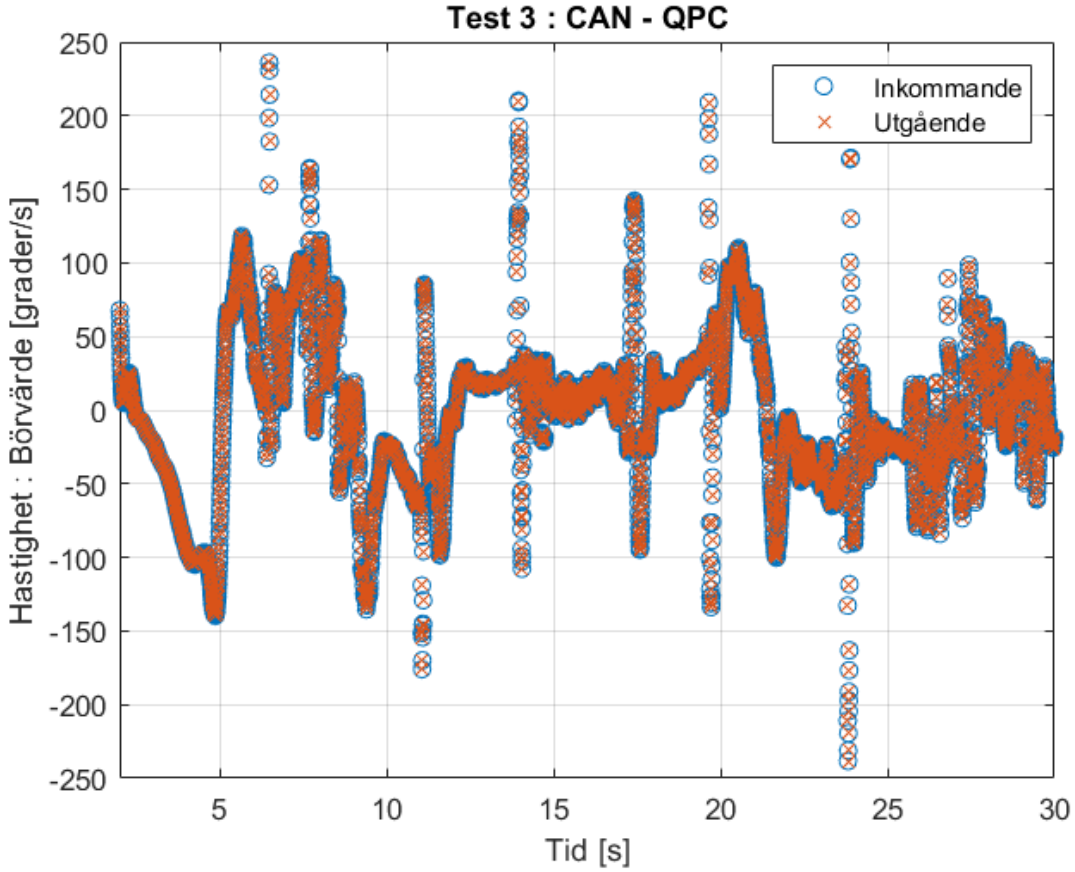
	Medelfördröjning [ms]	Medelfördröjning Enkel väg [ms]	Standardavvikelse [ms]
HIL	7 (Stickprov)	3,5 (Stickprov)	-
EtherCAT	6 (Stickprov)	3 (Stickprov)	-
CAN	2,126	1,063	0,789

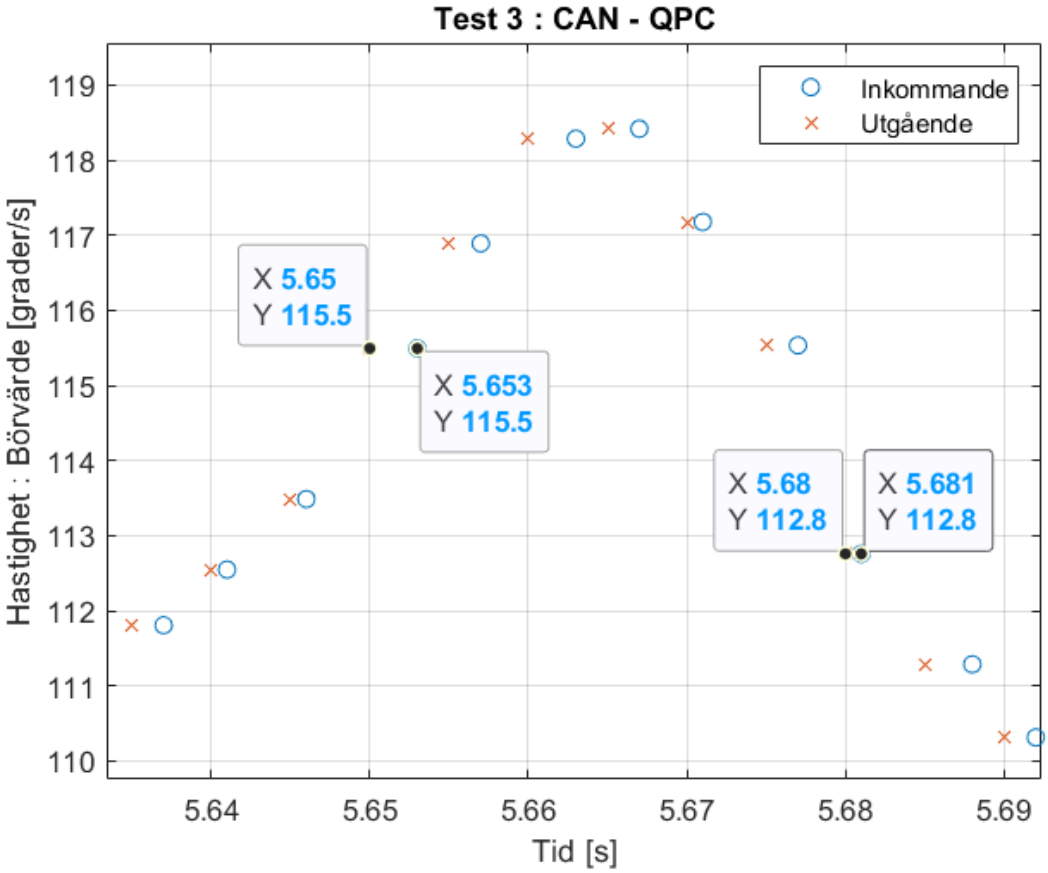



Test 3 : National Instruments - HIL
Inkommande
Utgående
X 6.236
Y 155.1
X 6.243
Y 155.1
X 6.256
Y 155
X 6.263
Y 155
Hastighet : Börvärde [grader/s]
Tid [s]
155.8
155.6
155.4
155.2
155
154.8
154.6
154.4
154.2
6.235
6.24
6.245
6.25
6.255
6.26
6.265
6.27


Test 3 : EtherCAT - QPC
Inkommande
Utgående
Hastighet : Börvärde [grader/s]
Tid [s]
250
200
150
100
50
0
-50
-100
-150
-200
-250
5
10
15
20
25
30


Test 3 : EtherCAT - QPC
Inkommande
Utgående
X 5.654
Y 116.9
X 5.661
Y 116.9
X 5.672
Y 116.4
X 5.679
Y 116.4
Hastighet : Börvärde [grader/s]
Tid [s]
119
118.5
118
117.5
117
116.5
116
115.5
115
114.5
5.65
5.655
5.66
5.665
5.67
5.675
5.68
5.685
5.69


Test 3 : CAN - QPC
Inkommande
Utgående
Hastighet : Börvärde [grader/s]
Tid [s]
250
200
150
100
50
0
-50
-100
-150
-200
-250
5
10
15
20
25
30

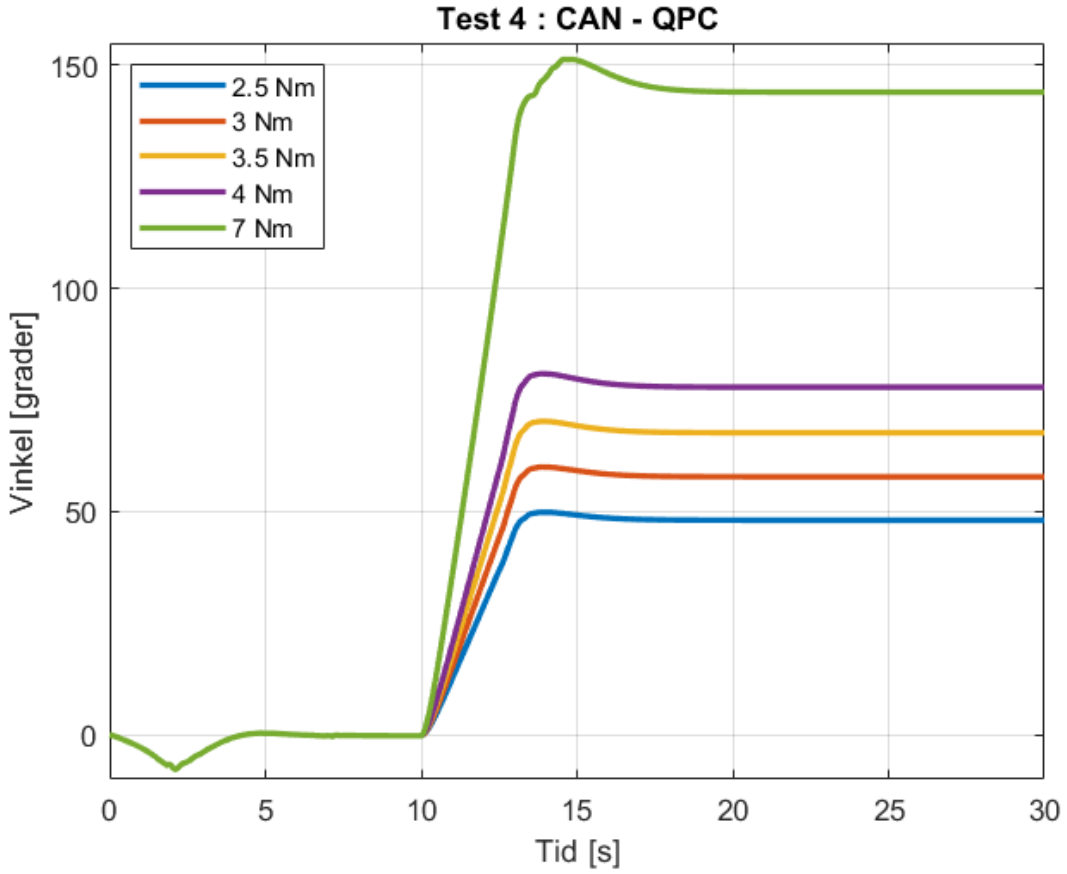

Test 3 : CAN - QPC
Inkommande
Utgående
X 5.65
Y 115.5
X 5.653
Y 115.5
X 5.68
Y 112.8
X 5.681
Y 112.8
Hastighet : Börvärde [grader/s]
Tid [s]
119
118
117
116
115
114
113
112
111
110
5.64
5.65
5.66
5.67
5.68
5.69

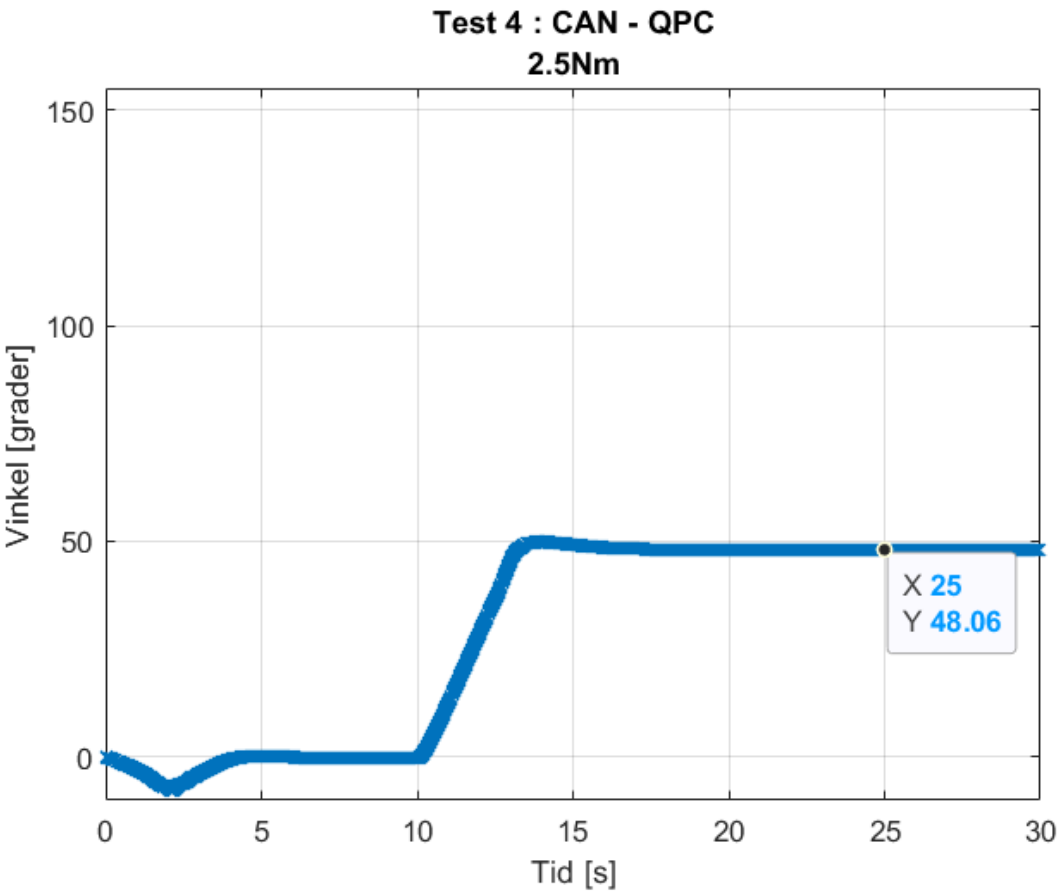
Test 4 - Grafer och data

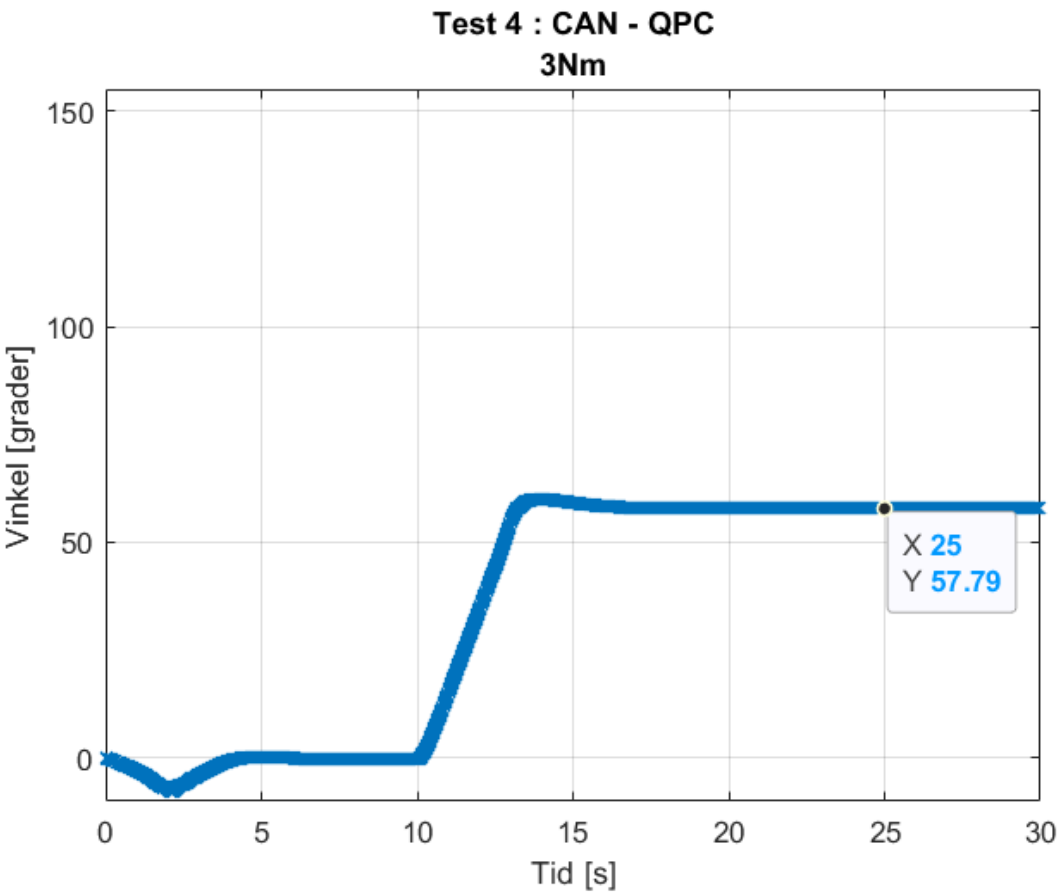
Graf över ändrad rattvinkel i CarMaker, vid ett pålagt moment av IPGdriver.
Matlab: plot(tid,vinkel)

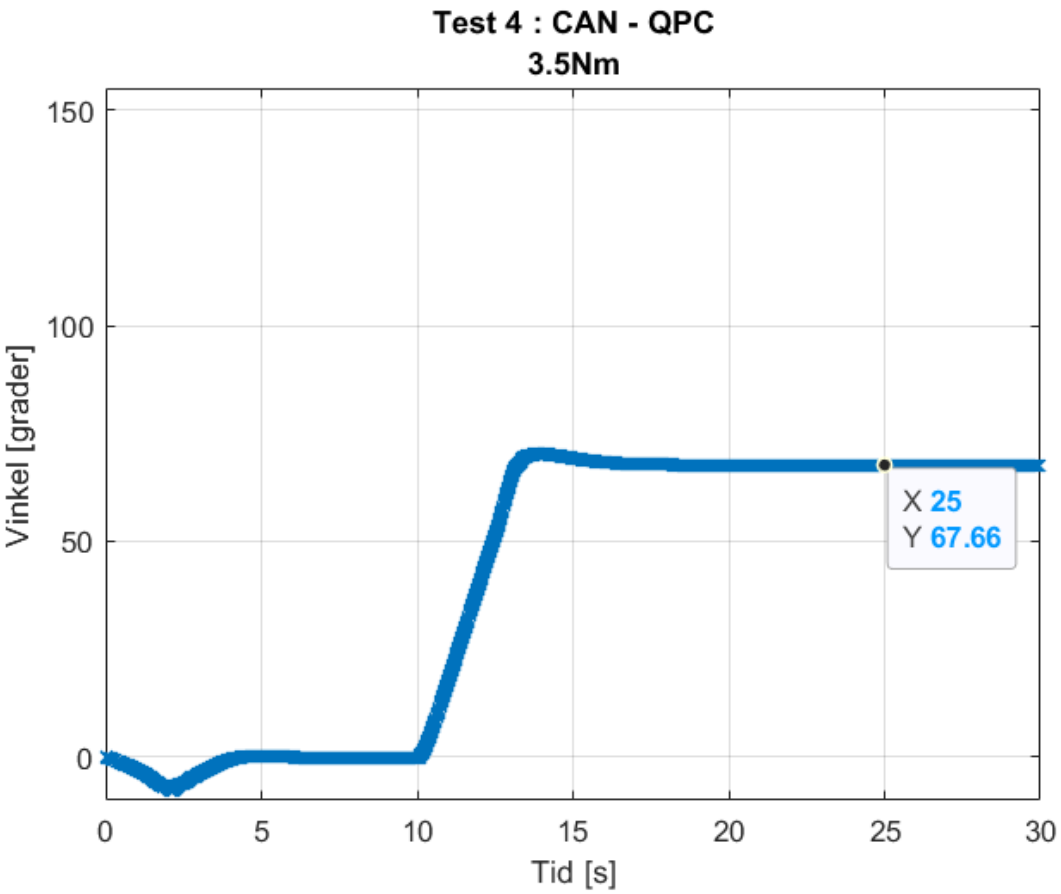
Tabell 4: Beräknad styvhet vid olika moment

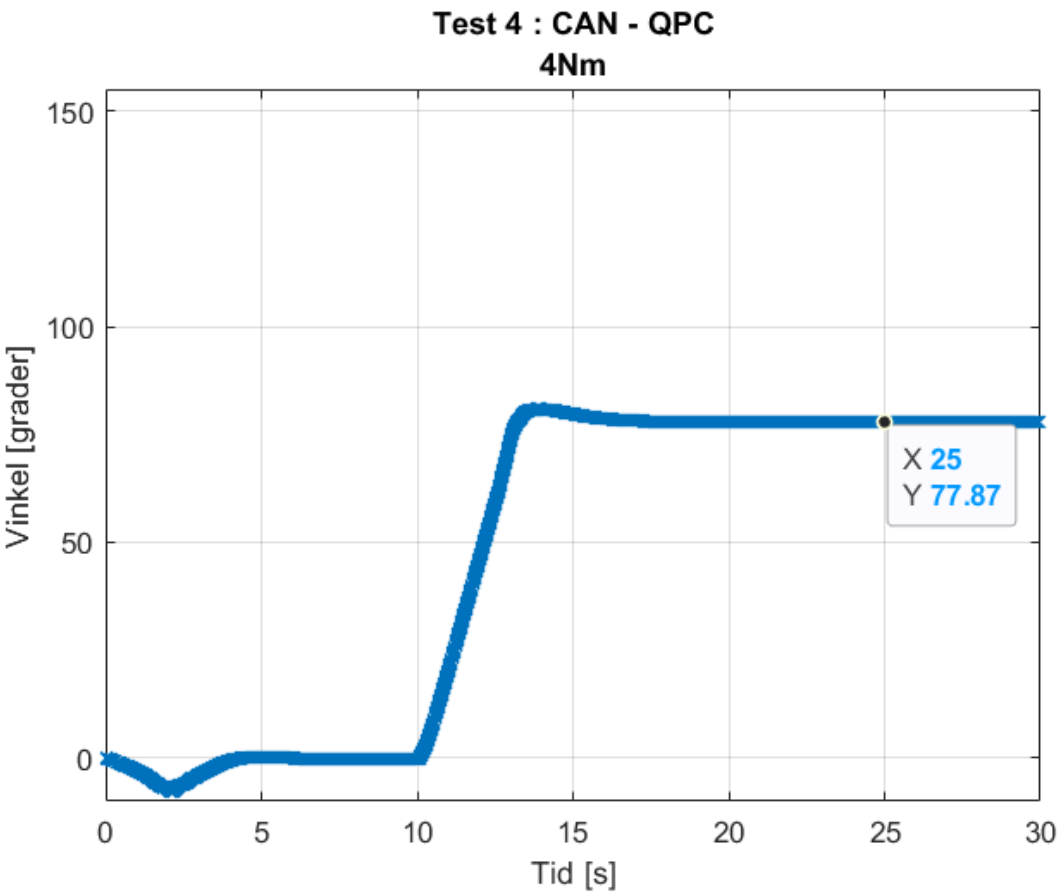
Moment [Nm]	Vinkel [grader]	Styvhet K_{CM} [Nm/grad]	Styvhet K_{RS} [Nm/grad]	Styvhet K_{TS} [Nm/grad]	Styvhet K_{SmH} [Nm/grad]
2,5	48,06°	0,052	12	2	0,054
3,0	57,79°	0,052	12	2	0,054
3,5	67,66°	0,052	12	2	0,054
4,0	77,87°	0,051	12	2	0,053
7,0	144°	0,049	12	2	0,050

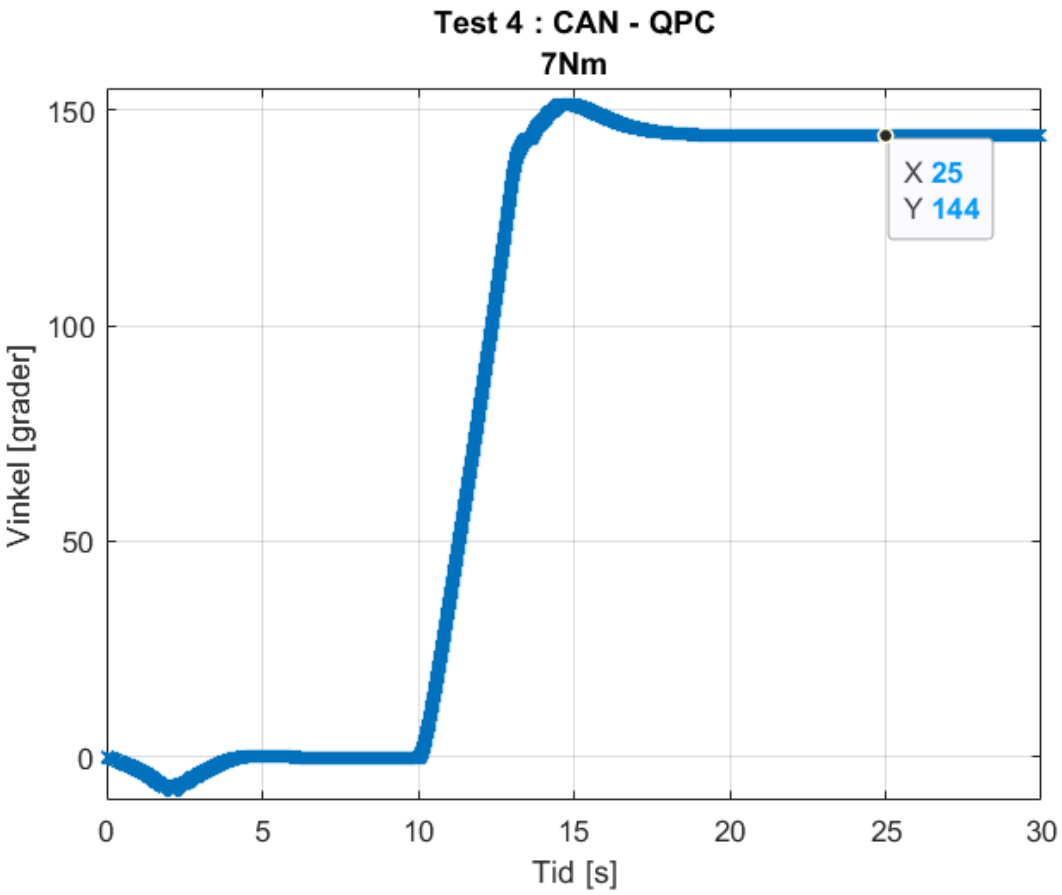



Test 4 : CAN - QPC
2.5Nm
Vinkel [grader]
Tid [s]
X 25
Y 48.06


Test 4 : CAN - QPC
3Nm
Vinkel [grader]
Tid [s]
X 25
Y 57.79

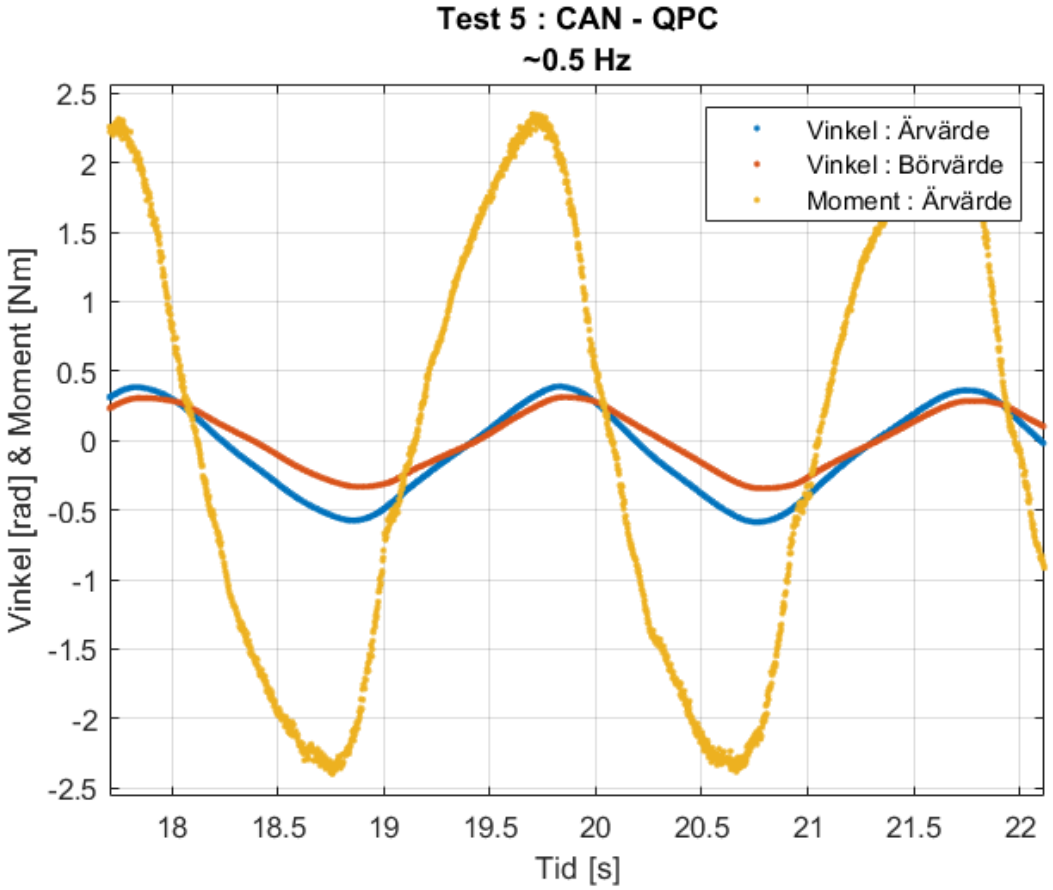

Test 4 : CAN - QPC
3.5Nm
Vinkel [grader]
Tid [s]
X 25
Y 67.66

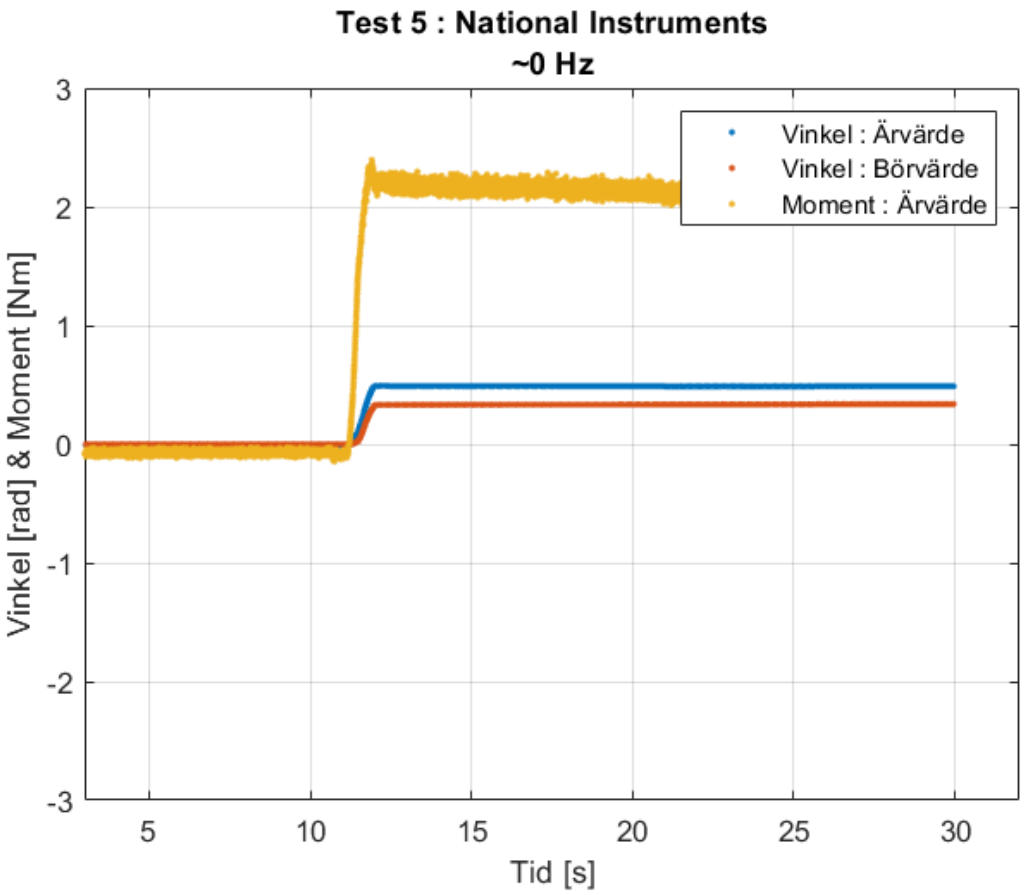

Test 4 : CAN - QPC
4Nm
Vinkel [grader]
Tid [s]
X 25
Y 77.87

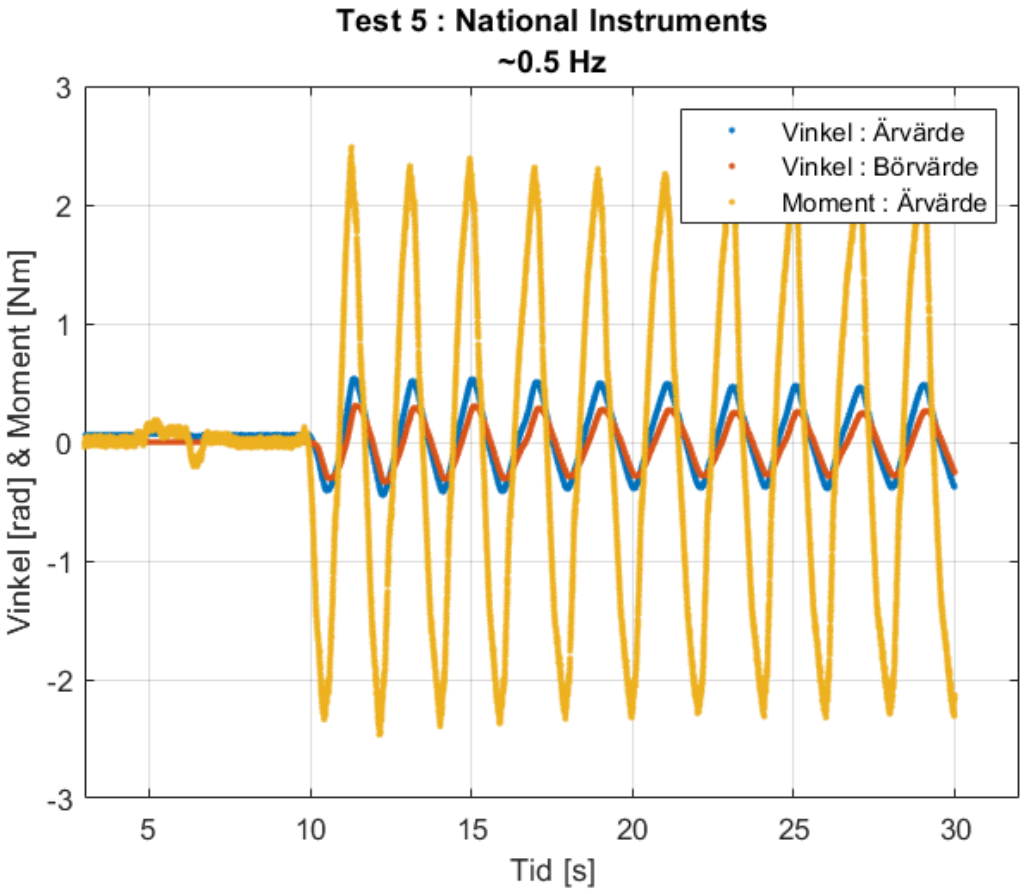

Test 4 : CAN - QPC
7Nm
Vinkel [grader]
Tid [s]
X 25
Y 144
150
100
50
0
0
5
10
15
20
25
30

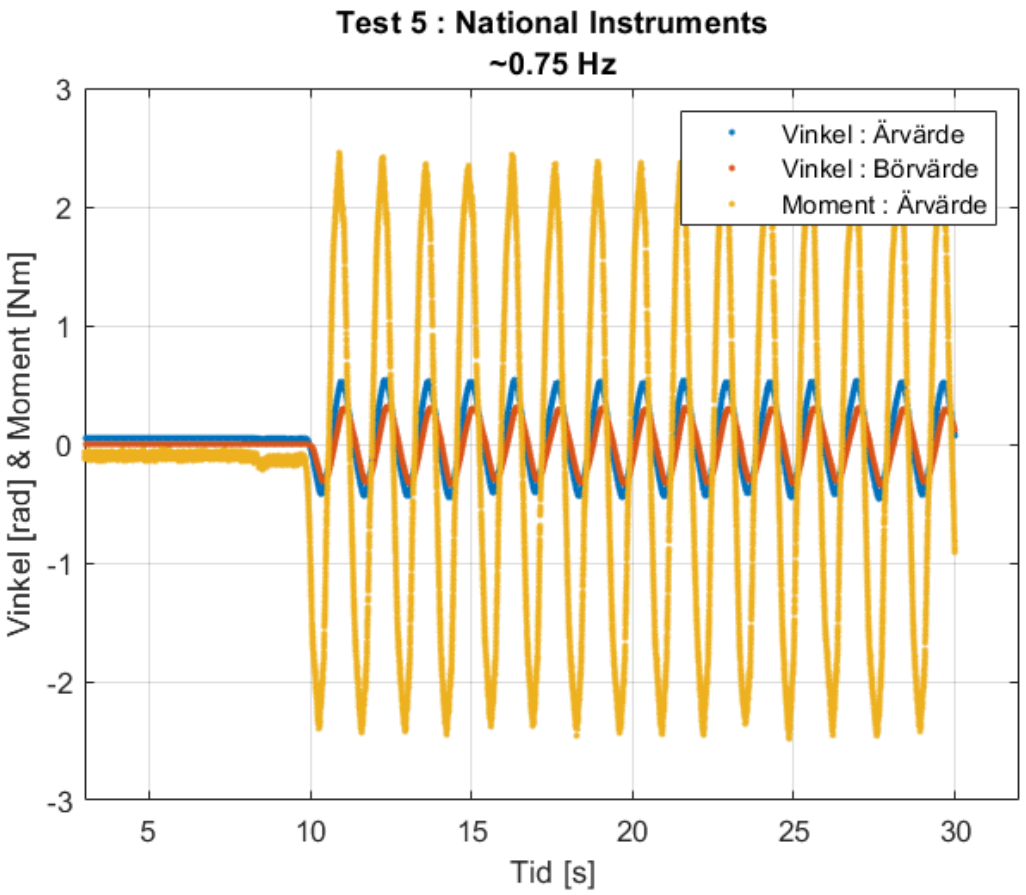
Test 5 - Grafer och data

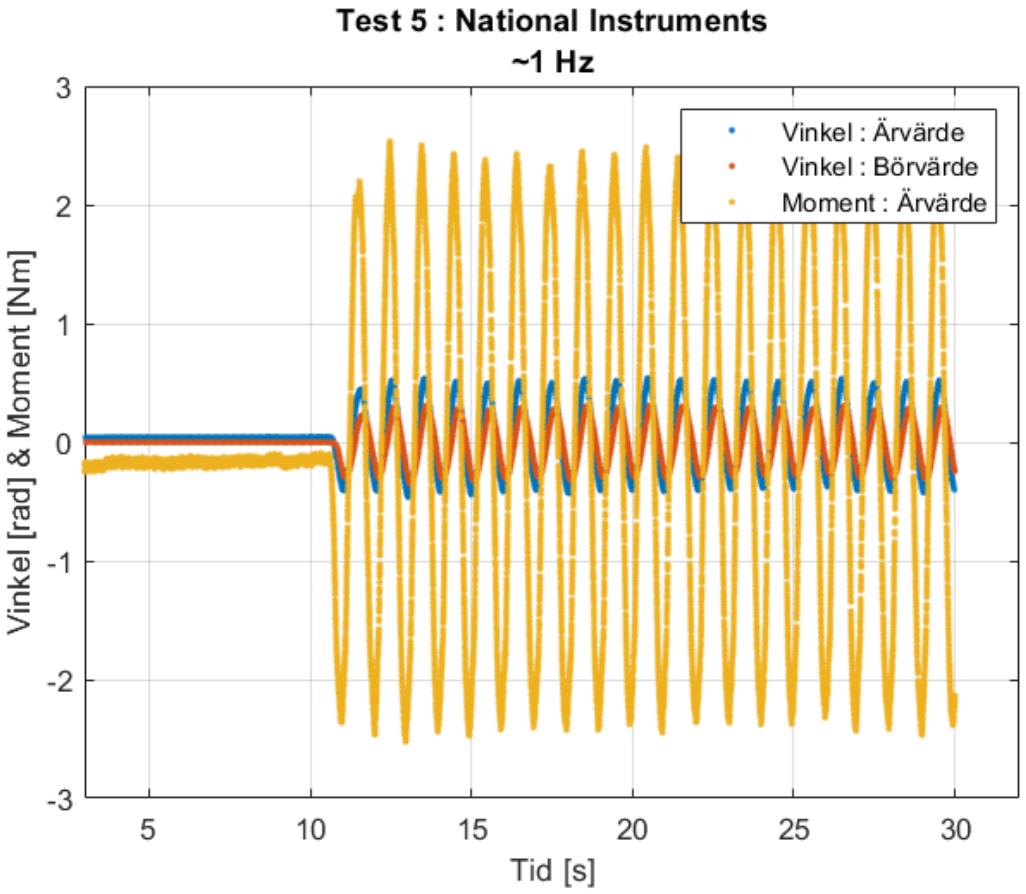
Börvärde och ärvärde av vinkel på den fysiska ratten(motorn). Även moment som lagts på den fysiska ratten av förare.
Matlab: plot(tid,vinkel ärvärde, tid,vinkel börvärde, tid,moment)
Graf över förstärkning och fasförskjutning.

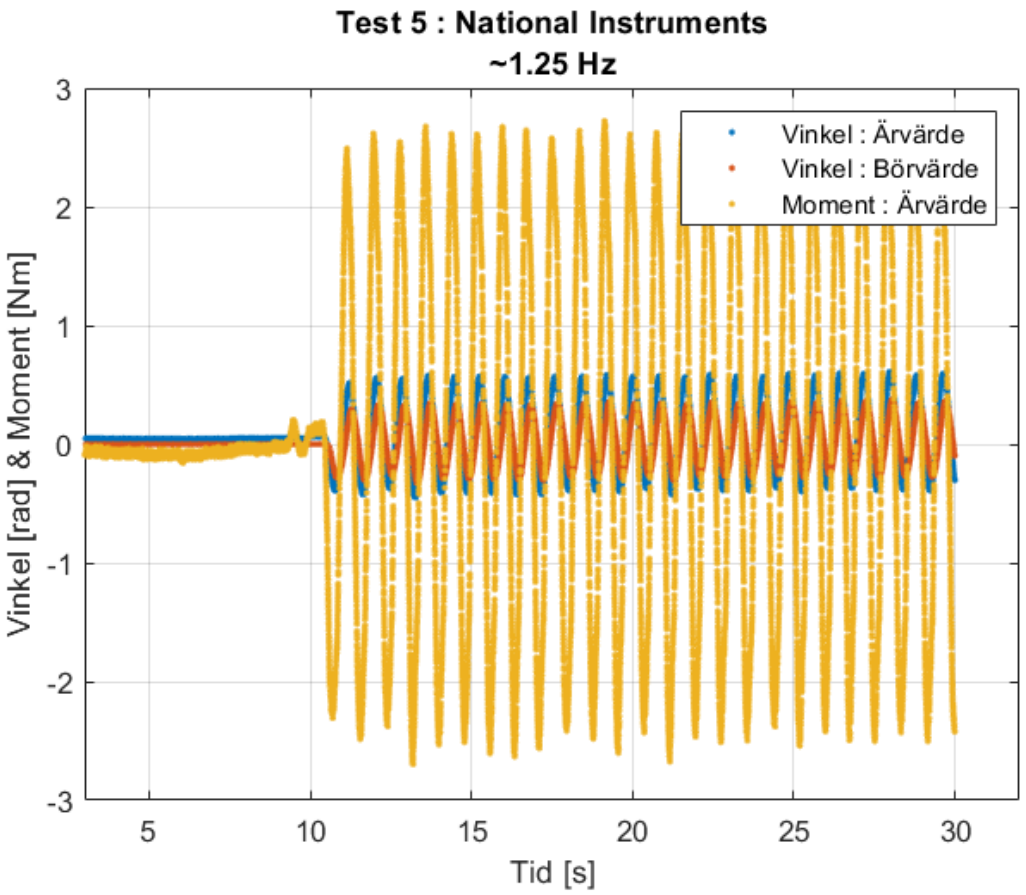


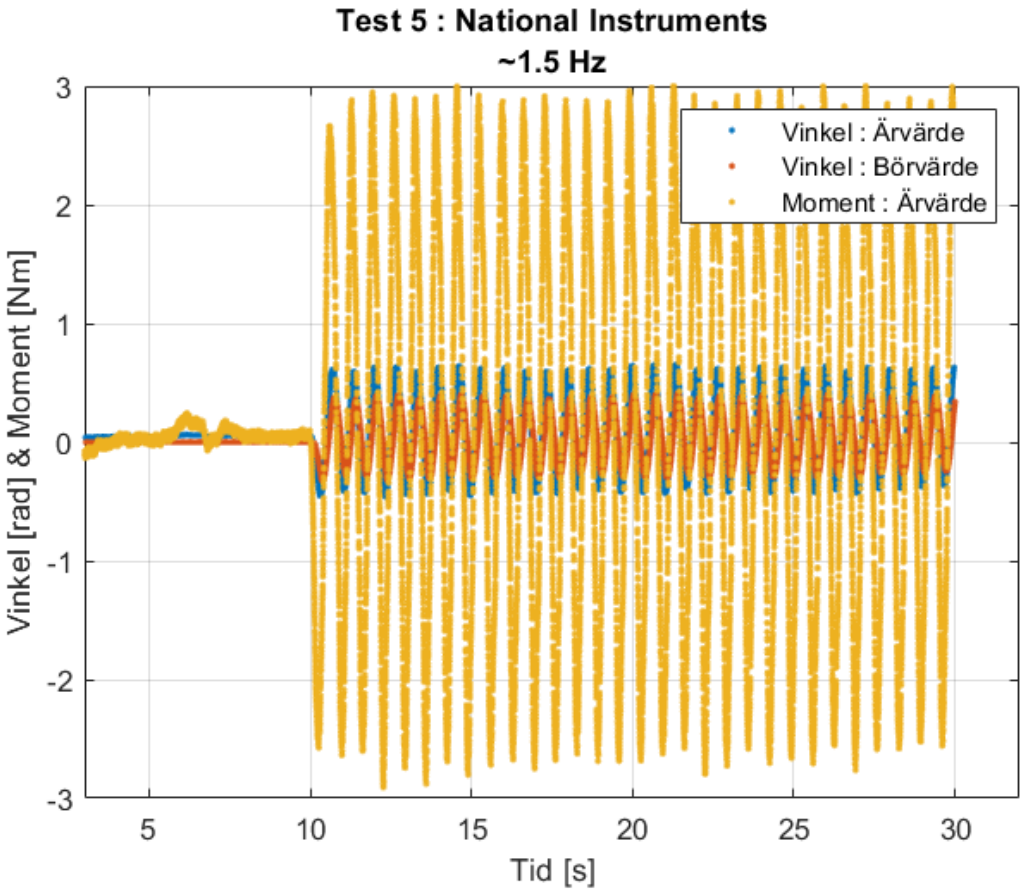

Test 5 : National Instruments
~0 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

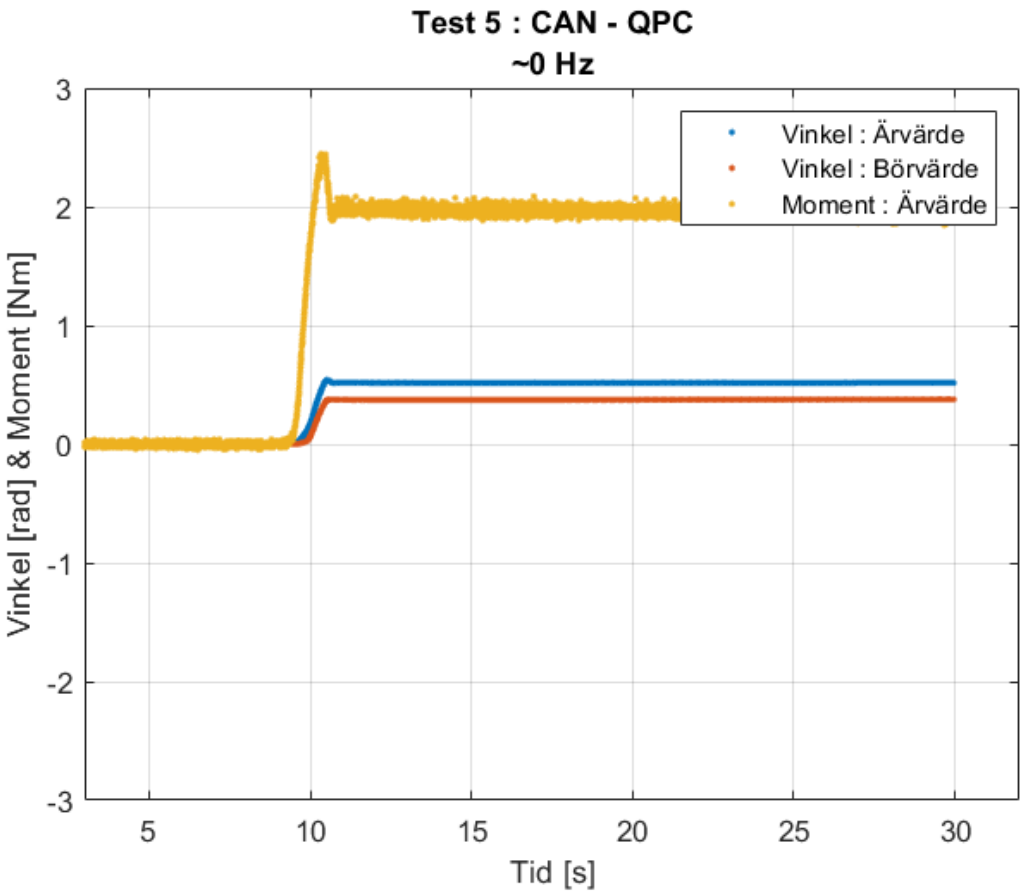

Test 5 : National Instruments
~0.5 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

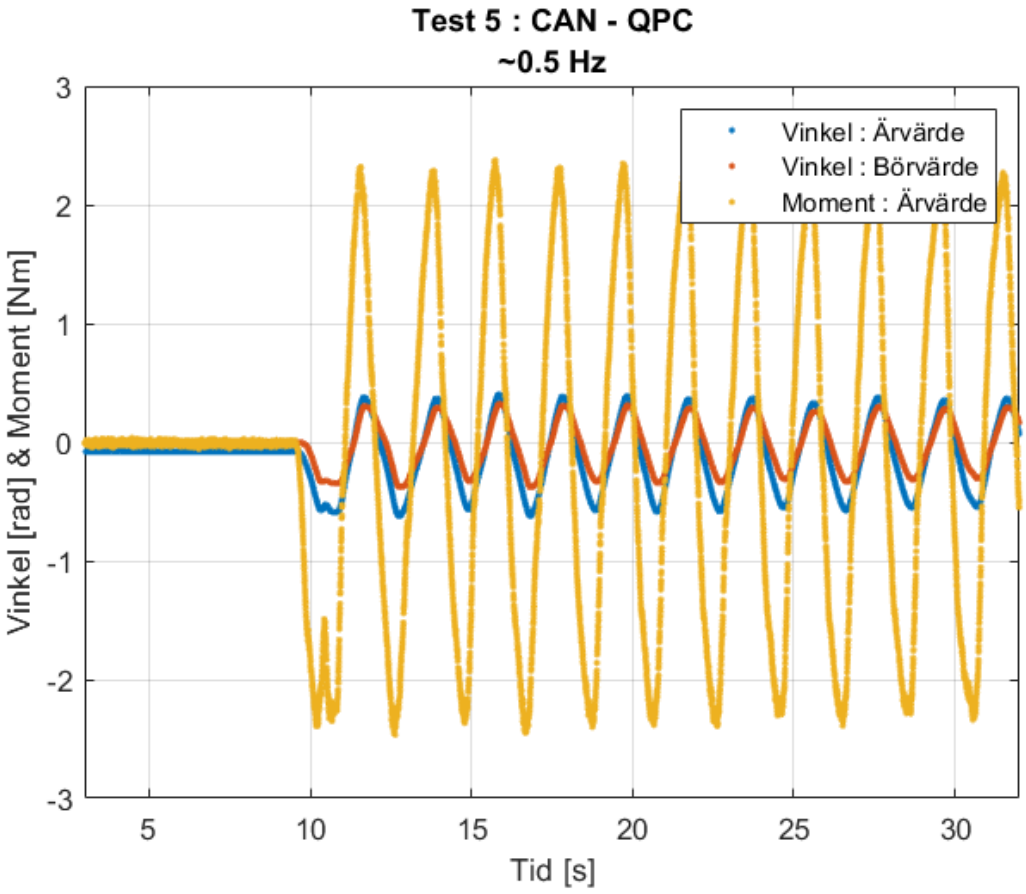

Test 5 : National Instruments
~0.75 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

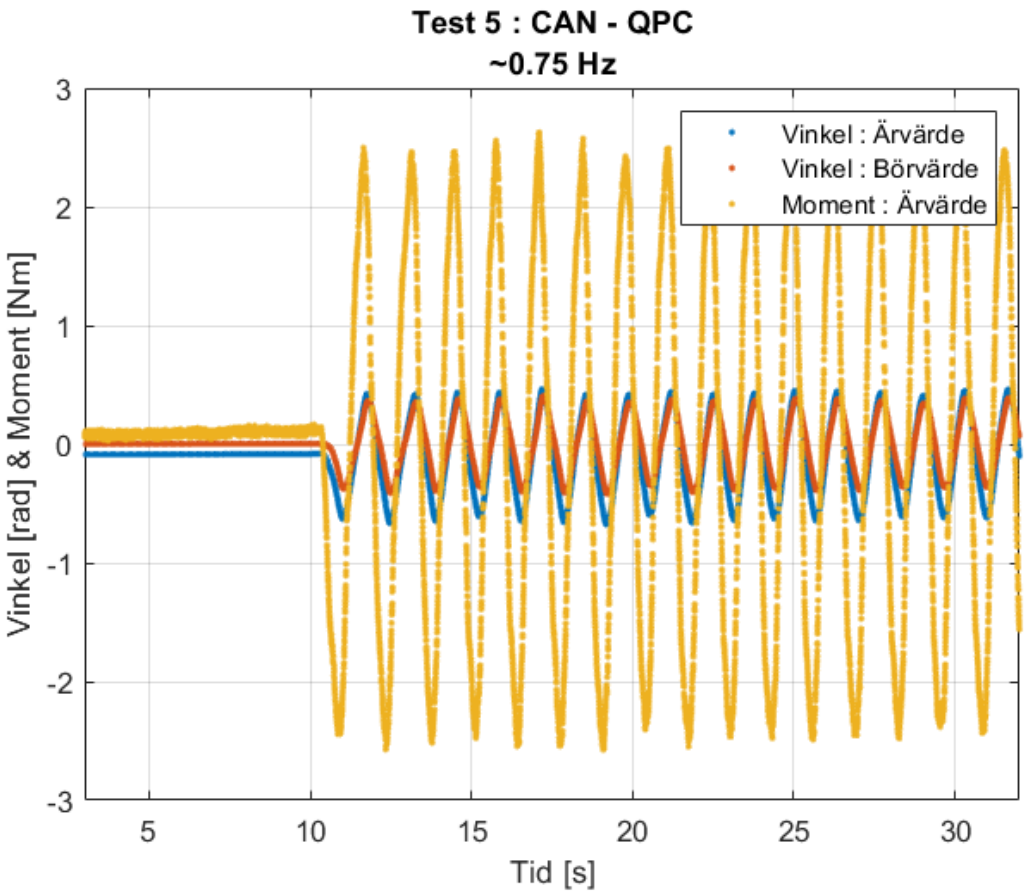

Test 5 : National Instruments
~1 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

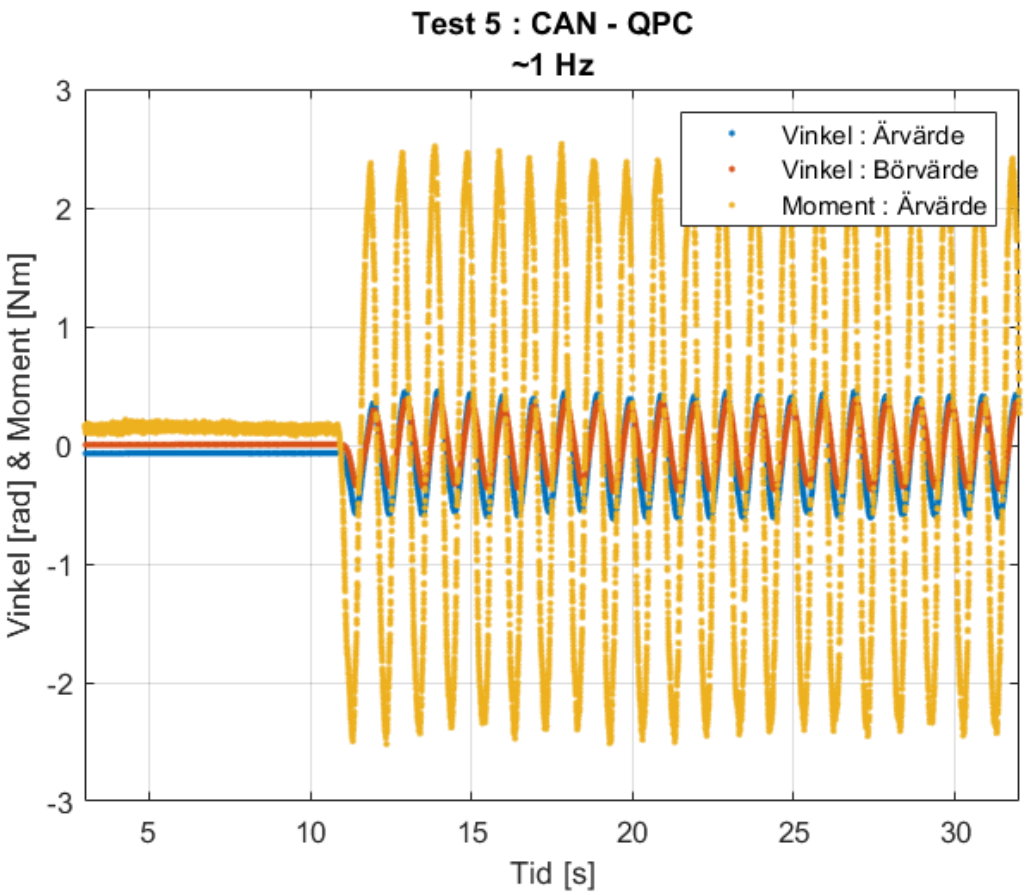

Test 5 : National Instruments
~1.25 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

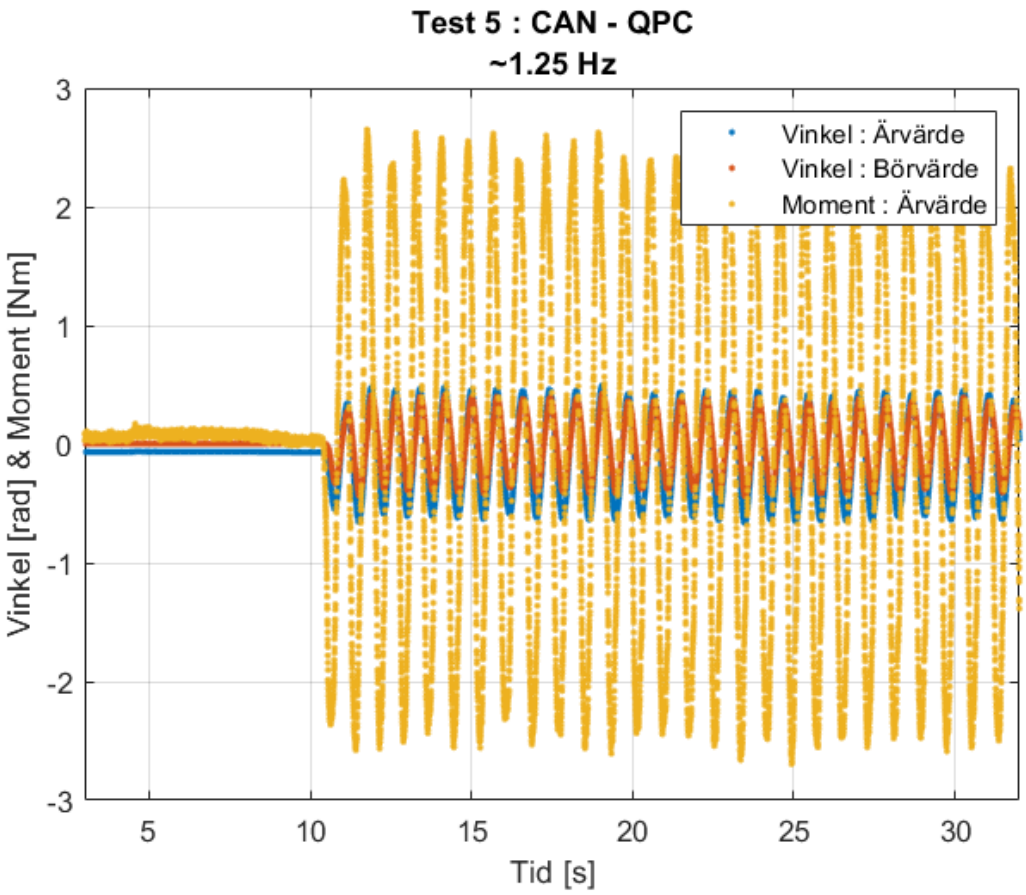

Test 5 : National Instruments
~1.5 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

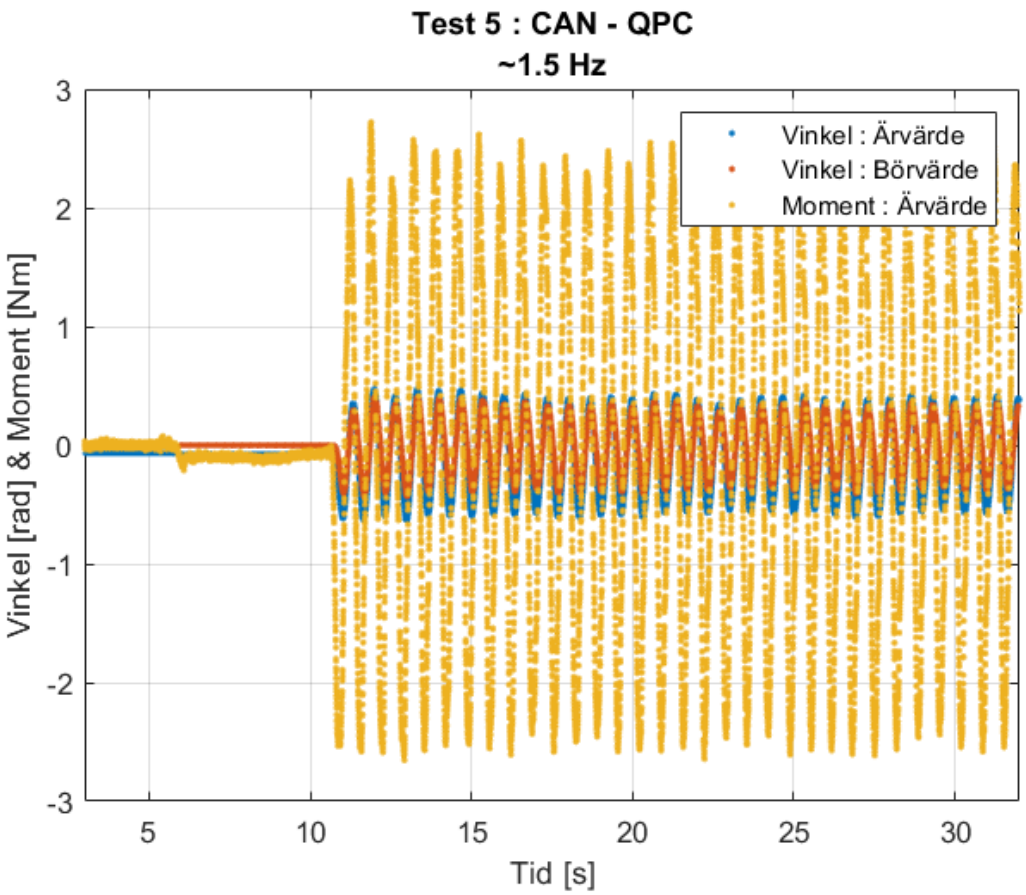

Test 5 : CAN - QPC
~0 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

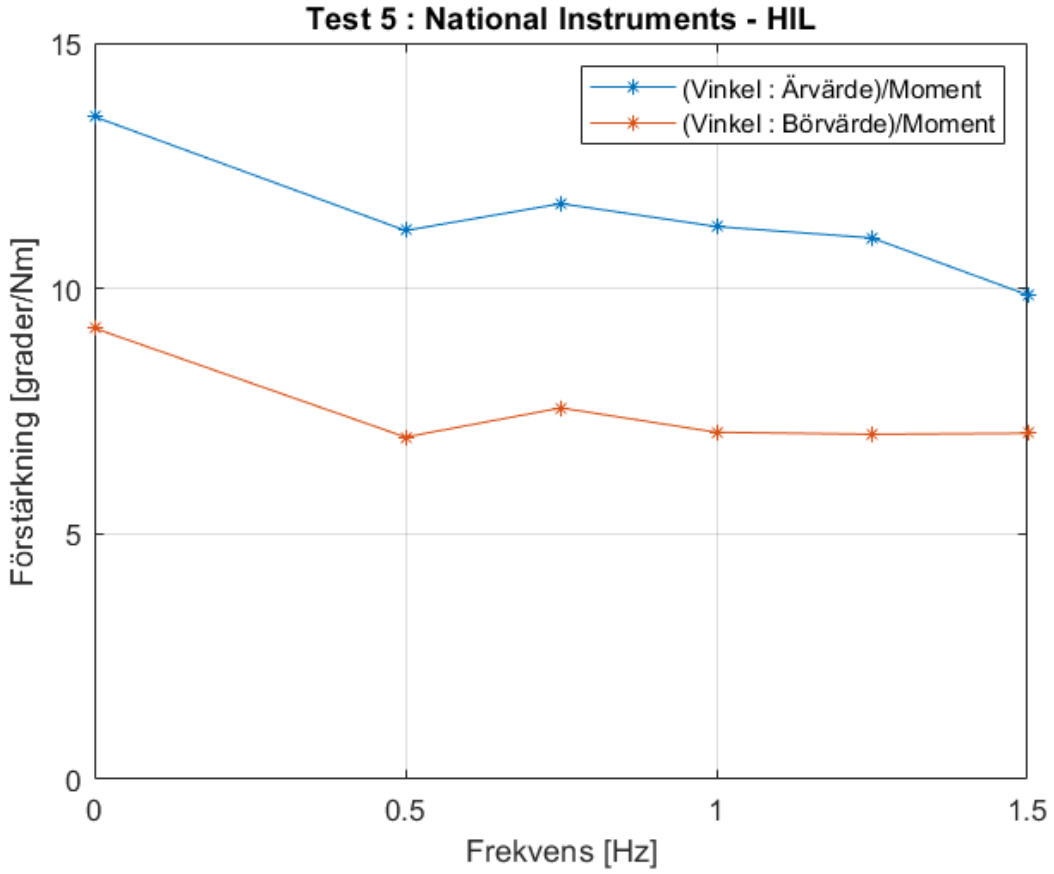

Test 5 : CAN - QPC
~0.5 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

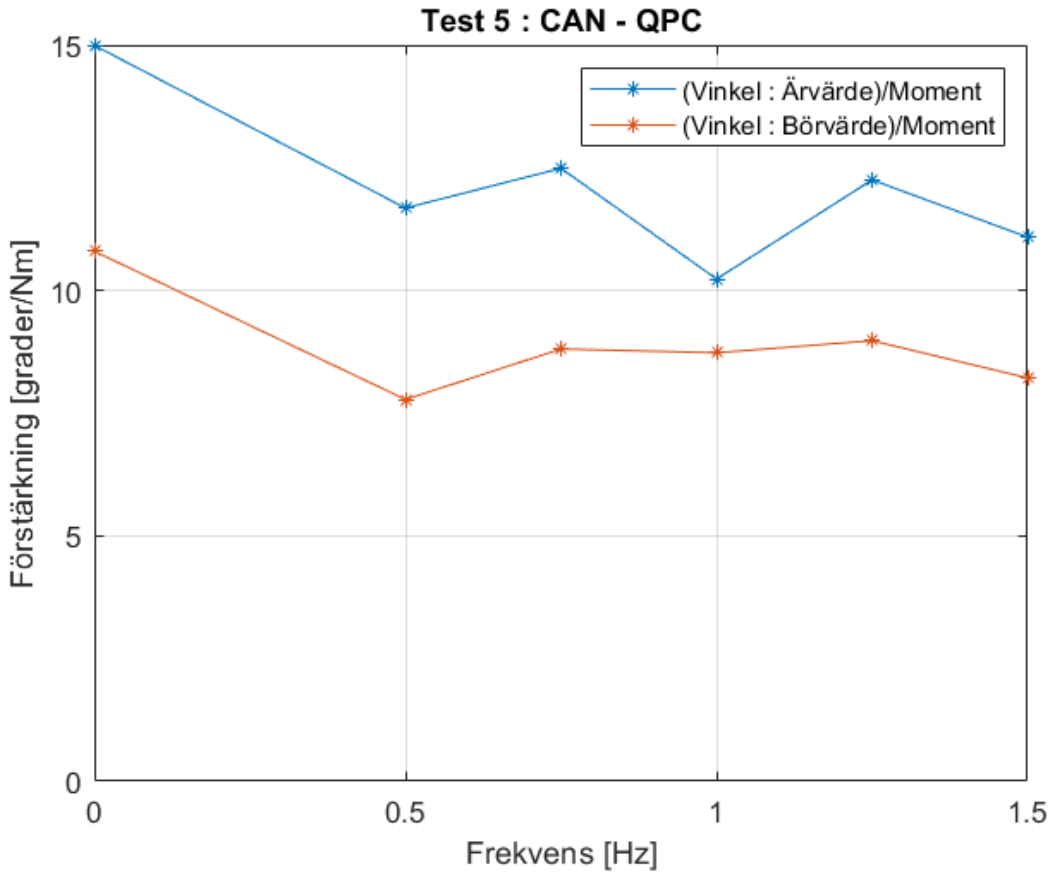

Test 5 : CAN - QPC
~0.75 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

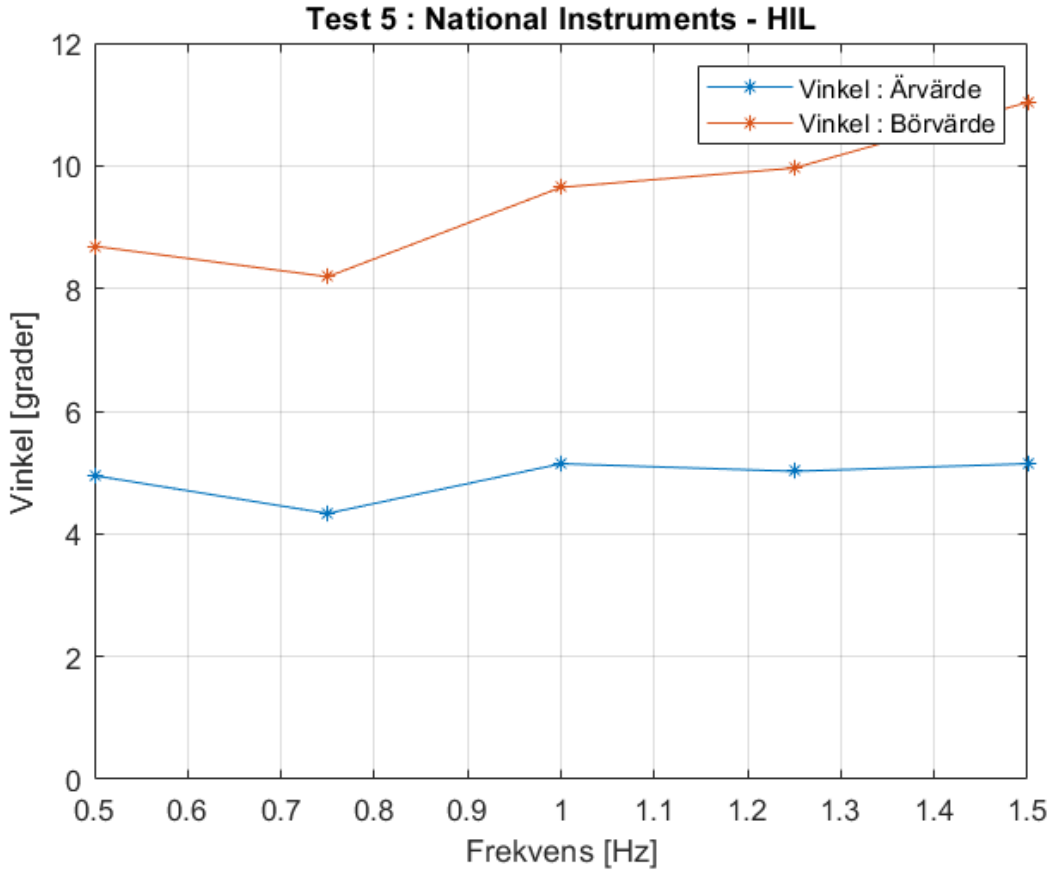

Test 5 : CAN - QPC
~1 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30

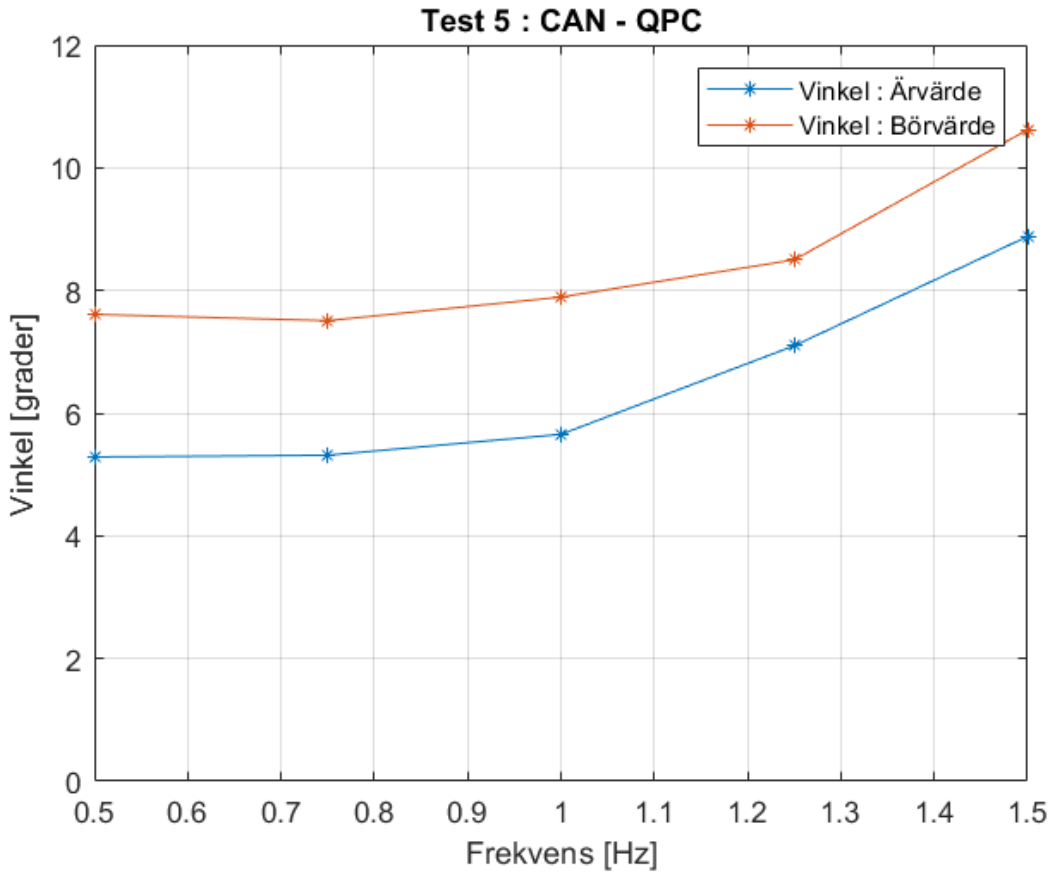

Test 5 : CAN - QPC
~1.25 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30


Test 5 : CAN - QPC
~1.5 Hz
Vinkel : Ärvärde
Vinkel : Börvärde
Moment : Ärvärde
Vinkel [rad] & Moment [Nm]
Tid [s]
3
2
1
0
-1
-2
-3
5
10
15
20
25
30


Test 5 : National Instruments - HIL
(Vinkel : Ärvärde)/Moment
(Vinkel : Börvärde)/Moment
Förstärkning [grader/Nm]
Frekvens [Hz]
0
5
10
15
0
0.5
1
1.5


Test 5 : CAN - QPC
(Vinkel : Ärvärde)/Moment
(Vinkel : Börvärde)/Moment
Förstärkning [grader/Nm]
Frekvens [Hz]
0
5
10
15
0
0.5
1
1.5


Test 5 : National Instruments - HIL
Vinkel : Ärvärde
Vinkel : Börvärde
Vinkel [grader]
Frekvens [Hz]
0
2
4
6
8
10
12
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5


Test 5 : CAN - QPC
Vinkel : Ärvärde
Vinkel : Börvärde
Vinkel [grader]
Frekvens [Hz]
0
2
4
6
8
10
12
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5

Test 6 - Enkätsvar och data

Svar av frågor vid test med förare.

Tabell 5: Medelvärde

	Fråga 1 Medelvärde	Fråga 2 Medelvärde	Fråga 3 Medelvärde	Fråga 4 Medelvärde	Fråga 5 Medelvärde	Fråga 6 Medelvärde
NI-HIL	4,25	3,5	3,5	4,75	4,25	1,5
EtherCAT	3,75	3,25	3	3,5	3,5	1
CAN	4	3	3	3,5	4	1

NI HIL

Förare: Anton Lundin — resultat 1-5

1. Övergripande känslan. [4,5]
2. Hur verkligt är upplevelsen jämfört med riktig bil? [4]
3. Hur väl flyter simuleringen/körningen (delay/lagg)? [3]
4. Hur känns trögheten i ratten jämfört med en riktig bil? [4,5]
5. “Känns” vägen i ratten? [4,5]
6. Hur åksjuk blev du? [2]
7. Övrig kommentar:

Förare: Jonas Sundberg — resultat 1-5

1. Övergripande känslan. [4]
2. Hur verkligt är upplevelsen jämfört med riktig bil? [3]
3. Hur väl flyter simuleringen/körningen (delay/lagg)? [4]
4. Hur känns trögheten i ratten jämfört med en riktig bil? [5]
5. “Känns” vägen i ratten? [4]
6. Hur åksjuk blev du? [1]
7. Övrig kommentar:

EtherCAT

Förare: Anton Lundin — resultat 1-5

1. Övergripande känslan. [4]
2. Hur verkligt är upplevelsen jämfört med riktig bil? [3]
3. Hur väl flyter simuleringen/körningen (delay/lagg)? [3]
4. Hur känns trögheten i ratten jämfört med en riktig bil? [3]
5. “Känns” vägen i ratten? [3]
6. Hur åksjuk blev du? [1]
7. Övrig kommentar:

Förare: Jonas Sundberg — resultat 1-5

1. Övergripande känslan. [3,5]
2. Hur verkligt är upplevelsen jämfört med riktig bil? [3,5]
3. Hur väl flyter simuleringen/körningen (delay/lagg)? [3]
4. Hur känns trögheten i ratten jämfört med en riktig bil? [4]
5. “Känns” vägen i ratten? [4]
6. Hur åksjuk blev du? [1]

7. Övrig kommentar:

CAN

Förare: Anton Lundin	resultat 1-5
1. Övergripande känslan.	[4]
2. Hur verkligt är upplevelsen jämfört med riktig bil?	[3]
3. Hur väl flyter simuleringen/körningen (delay/lagg)?	[3]
4. Hur känns trögheten i ratten jämfört med en riktig bil?	[3]
5. "Känns" vägen i ratten?	[4]
6. Hur åksjuk blev du?	[1]
7. Övrig kommentar:	

Förare: Jonas Sundberg	resultat 1-5
1. Övergripande känslan.	[4]
2. Hur verkligt är upplevelsen jämfört med riktig bil?	[3]
3. Hur väl flyter simuleringen/körningen (delay/lagg)?	[3]
4. Hur känns trögheten i ratten jämfört med en riktig bil?	[4]
5. "Känns" vägen i ratten?	[4]
6. Hur åksjuk blev du?	[1]
7. Övrig kommentar:	

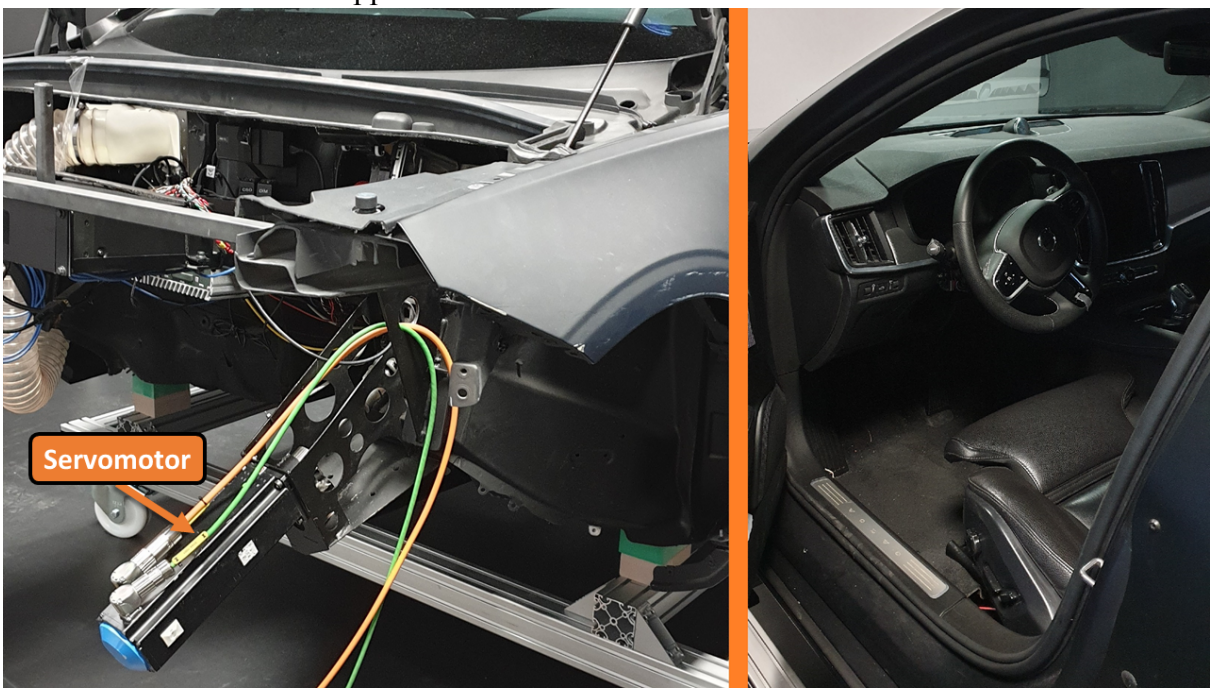
Bilaga 2 - Hårdvara och utrustning

Innehållsförteckning

Hårdvaran och Utrustning

Nedan följer den hårdvara och utrustning som används i projektet.

1. Servomotor med en axel kopplad till en bilratt



2. Styrenhet för styrning av servomotor och med kommunikationsmöjligheter över olika kommunikationsprotokoll som CAN och EtherCAT.

3. NI - HIL för simulering av CarMaker och med kommunikationsmöjligheter över t.ex. EtherCAT.

4. PC för simulering av CarMaker. Utrustad med ethernet för EtherCAT och USB för anslutning av CAN Interface. Även utrustad med FC1121 kort för EtherCAT anslutning.
   Processor: Intel Xeon
   Grafikkort: Nvidia Quadro RTX 4000
   Operativsystem: Windows 10 Enterprise

View basic information about your computer

Windows edition

Windows 10 Enterprise

© 2019 Microsoft Corporation. All rights reserved.

Windows 10

System

Processor:	Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz 3.60 GHz
Installed memory (RAM):	32,0 GB (31,7 GB usable)
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

5. Vector Interface VN1610. En USB-dongel så att en PC kan kommunicera över CAN.

6. Dongel för serial anslutning från PC till styrenhet via Cockpit.

7. Kablar, såsom Ethernet- och serialkablar.

8. Termineringsmotstånd 120 Ohm för CAN-bussen.

INSTITUTIONEN FÖR ELEKTROTEKNIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2021
www.chalmers.se

CHALMERS