CHALMERS

Car to infrastructure communication (C2I)

Examensarbete inom Data- och Informationsteknik

HENRIK ROSENBORG
CHRISTINA SCHOPPENBORG

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

EXAMENSARBETE

Car to infrastructure communication (C2I)

En kommunikationslösning mellan autonoma fordon och infrastruktur

HENRIK ROSENBORG
CHRISTINA SCHOPPENBORG

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg 2019

This page was left blank on purpose.

En kommunikationslösning mellan fordon och infrastruktur
Examensarbete inom Data- och Informationsteknik
HENRIK ROSENBORG
CHRISTINA SCHOPPENBORG

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA / Göteborgs Universitet
417 96 Göteborg
Telefon: 031-772 1000

Institutionen för Data- och Informationsteknik

Göteborg 2019

Sammanfattning

En av de mest omtalade ämnet i tekniska sammanhang är självkörande bilar. Det är ett kontroversiellt ämne och det kommer utan tvekan att medföra stora förändringar i våra trafiksystem. Innan självkörande bilar kan ta över vägarna måste de möta väldigt höga säkerhetsstandarder och rutiner. För att uppnå en säkrare och mer effektiv transport, måste fordon lära sig att förstå sin omgivning, till exempel infrastrukturen fordonet befinner sig i. Enligt det amerikanska transportdepartementet ligger den mänskliga faktorn bakom orsaken till olyckor i 94 % av fallen. [1] Infrastrukturen har inte utvecklats särskilt mycket de senaste åren medan bilarna blir mer moderna och även autonoma. För att hålla reda på de nya smarta bilarna måste infrastrukturen digitaliseras.

Rapporten beskriver en undersökning av möjligheterna för kommunikation mellan ett autonomt fordon och infrastruktur. Resultatet efter tester med företaget Infotivs autonoma bilplattform i kontorslandskap presenteras. Rapporten beskriver implementationen av bil till infrastrukturkommunikation bestående av en mjukvarulösning för kommunikation mellan fordon och infrastruktur samt demonstrations-enheter av en bro samt ett trafikljus.

Nyckelord: Autonoma fordon, IAP, ADAS, Python

Abstract

One of the most debated topics in tech nowadays is autonomous vehicles. It is a very controversial topic and it will undoubtedly bring major changes in our traffic systems in the future. According to the U.S Department of transportation the human error is the critical reason for 94 percent of vehicle crashes. [1] This results in many thousands of people who die in motor vehicle crashes every year around the world. Only in the United States more than 30,000 humans lost their lives in vehicle related accidents in 2015. Self-driving vehicles have the potential to reduce that number. Software could prove to be less error-prone than humans, but cybersecurity is still a chief concern.[2] Before autonomous cars can take over the roads, they must meet very high safety standards and routines. To achieve a safer and more efficient modern transportation, vehicles need to understand their surroundings such as the infrastructure it is located in. The infrastructure has not been further development while the cars are becoming more modern and autonomous. In order to keep up with the new smart cars, the infrastructure needs to be digitized. The thesis describes an examination of the possibilities of communication between an autonomous vehicle and infrastructure. The result after tests with the company Infotiv's autonomous car platform in office landscape is presented. The report describes an implementation of car for infrastructure communication consisting of a software solution for communication between vehicles and infrastructure as well as demonstration units of a bridge and a traffic light.

Förord

Denna rapport är ett examensarbete inom mjukvaruutveckling på mekatronikprogrammet vid Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola. Examensarbetet utfördes på teknikföretaget Infotiv AB och beskriver utvecklingen av en mjukvarulösning för kommunikation mellan fordon och infrastruktur.

Särskilt tack riktas till:

Joachim von Hacht, handledare på CTH, för vägledning under examensarbetets genomförande.

Nils Gangby, handledare på Infotiv AB, för vägledning under den tekniska utvecklingen.

Terminologi

C2I- Kort för Car to Infrastructure som är en benämning på system för kommunikation mellan fordon och infrastruktur.

IAP- Infotiv Autonoma Plattform som är ett styrsystemplattform byggt som en modell av ett fordon.

ADAS - Kort för Advanced Driver Assistance System, används för att validera alla signaler innan dessa går till motorn.

Python- Python är ett programmeringsspråk som användes för mjukvarulösningen.

***Socke*t** - Typ av virtuell kontakt, vilken dataöverföring kan ske.

Publisher Socket - En virtuell kontakt som är inställd att vara utgivaren av data.

Subscriber - En virtuell kontakt som är inställd att vara prenumerant av data.

Serialisering - En process som transformerar en datastruktur eller ett objekttillstånd, till ett format som går att lagra och överföra.

Avserialisering - Serialiserad data som återskapas till sitt ursprungstillstånd.

Innehållsförteckning

1. Bakgrund

Vi bevittnar en storskalig ökning och användning av automatiserade fordon i olika miljöer. I dagsläget används så kallade Automated Guided Vehicles (AGV) i många industrier och välkända företag. AGV system består av automatiserade fordon som följer markeringar. Dessa används för transport av t.ex. material. AGVs är ett exempel på automatiserade fordon som är kapabla att köra autonomt med hjälp av vägledning. [3]
Det finns även automatiserade fordon som till exempel automatiserade lastvagnar som kan röra sig mer självständigt i t.ex. gruvor från punkt A till punkt B och vice versa.

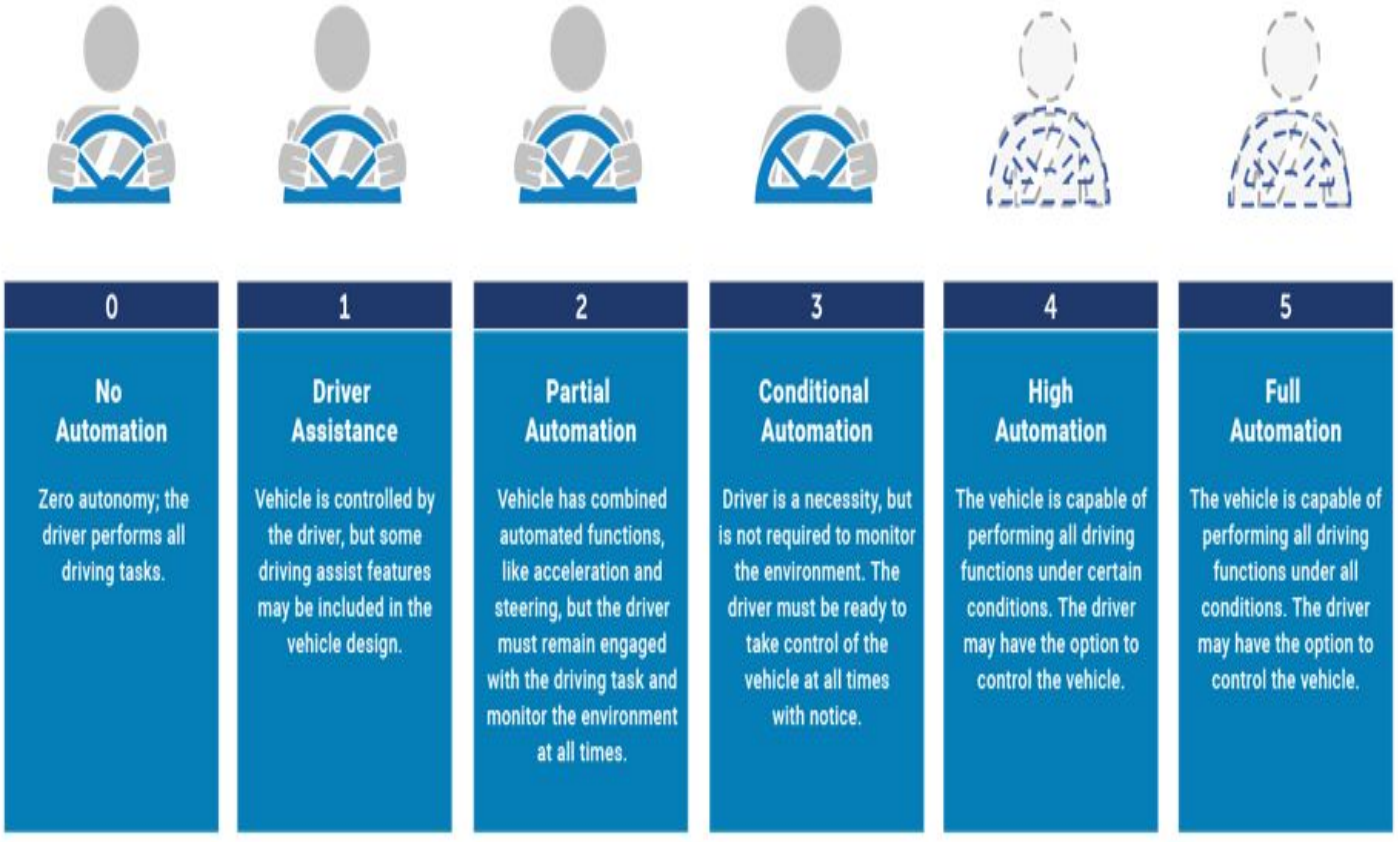
Det har skett stora framsteg i bilindustrin de senaste åren. Nya system för automatisering har utvecklats och implementerats i moderna fordon för att underlätta körningen. Nästa steg kan vara att eliminera den mänskliga faktorn helt och ersätta det med autonoma system. Sådana system finns redan i test i så kallade självkörande fordon, ofta refererade till som autonoma fordon. I autonoma fordon behövs ingen förare som har kontroll över fordonet för att fordonet skall kunna köra säkert. Under testning används dock normalt en förare.

Autonoma fordon i trafiken måste ta samma logiska beslut som människor idag tar. Vid exempelvis en korsning behöver fordonet känna till när det skall köra och när det skall stanna. För att detta skall vara möjligt måste fordonen ha tillgång till samma information som människor använder för att basera deras beslut på.
En lösning för att ge fordonet tillgång till denna sorts information är att upprätta en kommunikation med en omgivande infrastruktur, till exempel trafikljus, vägskyltar, broar etc. Detta kräver att fordon skall kunna koppla upp sig till andra enheter och fordon. Sådana fordon refereras till som uppkopplade fordon. Inom fordonsindustrin benämns i tekniken C2X eller V2X (Car To X, respektive Vehicle To X).
Vi kommer använda begreppet ***C2I (Car to Infrastructure)*** för detta examensarbete eftersom detta begrepp används av företaget som detta examensarbete utfördes på. C2I motsvarar V2I. Se Figur 1.1.

Figur 1.1: Illustration av olika möjliga uppkopplingar av autonoma fordon. Man kan koppla upp fordon till infrastruktur, fordon till fordon, fordon till moln. Man kan också etablera kommunikation mellan fordon och fotgängare eller kommunikation mellan fordon och allt annat som man kan etablera en kommunikation med. [4]

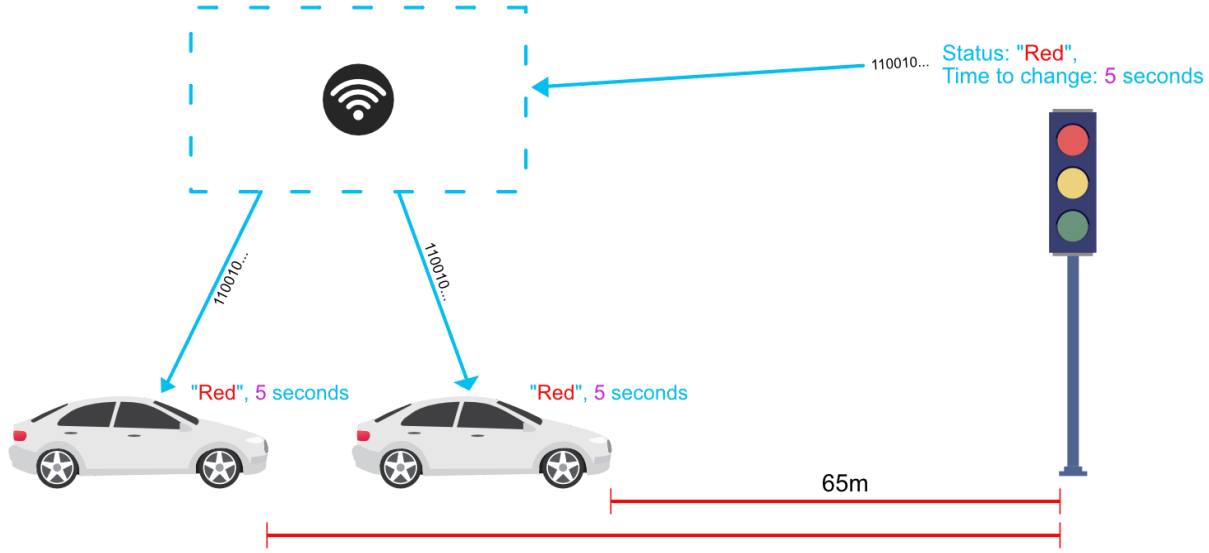
Uppkopplade autonoma fordon använder olika kommunikationstekniker för att kommunicera med till exempel föraren, andra fordon på vägen och infrastrukturen i sin omgivning. Dessa tekniker används och utvecklas i syfte att ständigt förbättra säkerheten under körning men också effektivisering av trafikflöden och optimering av körtid.

Autonoma fordon kan indelas i 6 olika nivåer då olika fordon är kapabla av autonom körning i olika grad, som oftast beskrivs av forskare på en skala från 0 till 5 enligt Figur 2 nedan. På de amerikanska vägarna definieras autonoma fordon av US Department of Transportation National Highway Traffic Safety Administration (NHTSA). De har gjort flera definitioner för olika nivåer av automatisering. Detta har gjorts för standardiseringens skull och för att underlätta tydlighet och konsistens. NHTSA har antagit SAE: s internationella definitioner för automatiseringsnivåer. Dessa definitioner delar upp fordon i nivåer baserade på "vem gör vad och när." [5]. Se Figur 1.2.



Figur 1.2: Illustrationen visar olika nivåer som självkörande bilar kan klassificeras i. Bilden visar hur NHTSA har definierat och delat upp autonomitet i bilar. [6]

Möjligheterna med C2I sträcker sig längre än vad människor kunnat använda samma teknik till. Vid kommunikation med ett trafikljus exempelvis kan fordonet ta emot data trådlöst långt före trafikljuset är synligt. Datan som skickas via C2I kan dessutom göras mera utförlig med till exempel information om tid tills nästa statusbyte. Fordonet skulle kunna ta ett beslut om vilken hastighet det skall hålla för att anlända vid trafikljuset när det är grönt. Detta ökar miljövänlig körning, säkerheten i trafiken och effektiviteten av trafikflödet. En illustration av detta visas i Figur 1.3.



Figur 1.3: Visar hur kommunikationen skulle kunna ske mellan ett autonomt fordon och ett trafikljus.

1.2 Infotiv

Detta arbete genomfördes på förfrågan av Infotiv. Infotiv är ett teknikföretag i Göteborg som bedriver verksamhet inom olika områden som medicinsk teknik, Life Science, kärnkraft samt inbyggda system med fokus på fordonsindustrin.
På Infotiv utvecklades en bilplattform för att efterlikna kommunikationen i en vanlig Volvo-bil. Bilplattformen skapades med syftet att användas för utbildning internt på företaget och för att möjliggöra forskning och utveckling av ny teknik för autonoma fordon.

1.3 Syfte

Syftet med examensarbetet är att utveckla ett koncepttest för att undersöka och testa möjligheterna för kommunikation mellan en autonom bil och omgivande infrastruktur. Bilen skall utifrån informationen den mottar från infrastrukturen kunna interagera och ta beslut på liknande sätt som människor agerar med infrastruktur.

1.4 Mål

Målet är att utveckla ett prototyp-system för en modell av kommunikation mellan en bil och en infrastruktur i form av ett trafikljus och en bro som skall simulera broöppning eller brostängning. Detta innefattar att utveckla nödvändig hårdvara för att simulera infrastruktur i form av ett trafikljus och en bro samt att ta fram mjukvara som möjliggör att infrastrukturen kan publicera data via ett nätverk. Bilen skall kunna ta emot och agera enligt informationen i den skickade datan från trafikljuset och bron.

Bilen skall kommunicera med trafikljuset för att avgöra om farten bör sänkas eller ökas innan bilen har nått trafikljuset för att optimera körningen.

För brobeslut är tanken att bilen skall kommunicera med signalerna från bron för att avgöra om farten bör minskas eller ökas, eller om man skall ta en omväg istället för att minska försening.

1.5 Avgränsningar

I brist av tid kommer bilen endast att kommunicera med två typer av infrastruktur. Infrastrukturen kommer att bestå av ett trafikljus och en bro. Bron kommer att användas för broöppning och för att bilen ska kunna ta brobeslut medan trafikljuset kommer att användas för att bilen ska ta ljussignalbeslut.

1.6 Metod

Befintlig bilmodell av autonom bil fanns redan vid examensarbetets start. Bilen är kapabel att kommunicera med en annan autonom bilmodell. Första steget blev att studera bilen för att få en förståelse för hur kommunikationen skulle kunna gå till. Instudering och förarbete gjordes dessutom kring kommunikation. Webbsökning på tidigare implementationer gjordes dessutom för att förstå möjligheterna med tekniken.

I andra steget planerades att skapa infrastrukturen, så att bilmodellen kunde testas mot något. Den nödvändiga infrastrukturen i projektet utgörs av ett trafikljus och en bro. Trafikljuset och bron har konstruerats med hjälp av olika hårdvarukomponenter så som en Raspberry Pi dator och dioder som styrs med en Raspberry Pi datorn i syfte att efterlikna trafikljus.

I sista steget utvecklades mjukvarumoduler med olika funktionaliteter. Arbetet genomfördes med agil metod och delades upp i delmål enligt Scrum-schemat som togs fram under planeringen.

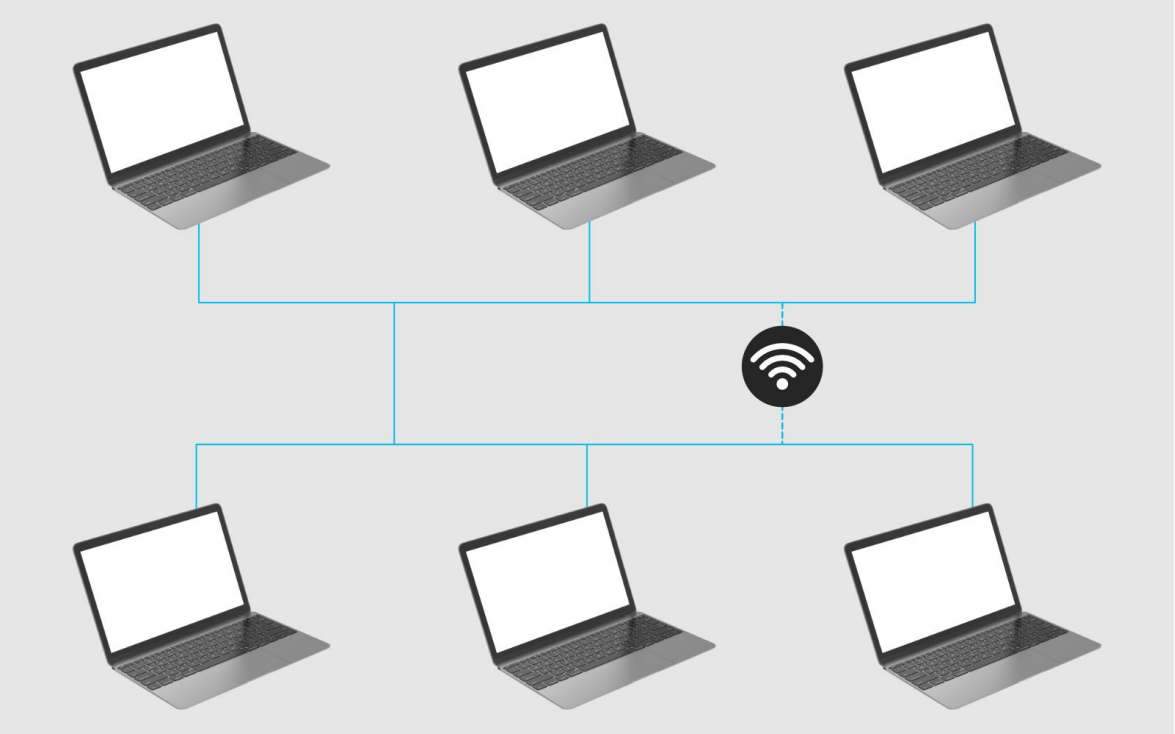
Ordningen i vilken mjukvaran utvecklades blev naturlig utefter hur grundläggande dess funktionalitet var för projektet. Den första modulen, hanterar exempelvis inhämtningen av data från infrastrukturen. Varje modul utvecklades tills dess funktionalitet var fullständig innan utvecklingen av nästa modul med ny funktionalitet påbörjades.

2. Teoretisk bakgrund

Den teoretiska kunskap och koncept som ligger till grund för detta examensarbete.

2.1 Datornätverk

Ett datornätverk består av enheter som utbyter data med varandra genom specificerade protokoll, såsom IP, TCP, TCP/IP eller HTTPS. Protokollen innehåller regler för hur överföringen skall gå till. [7]



Figur 2.1: Illustration av ett datornätverk [8]

2.1.1 IP

Ett protokoll för överföring av data via internet. Med IP överförs data i små paket mellan avsändare och mottagare som har varsin IP-adress. Adresserna består av siffror och punkter, exempelvis *37.456.78.93*. Dessa adresser används på samma sätt som gatuadresser används för att dirigera post, fast digitalt mellan maskiner. Informationen kan ta många vägar fram till mottagaren inom datornätverket, dock garanteras inte att informationen kommer fram. Protokollet kan användas på olika sätt, beroende på vilket protokoll det kombineras med i transportskiktet. [9]

Figur 2.2: Illustration av internetprotokollet [10]

2.1.2 TCP

Ett dataöverföringsprotokoll som bygger vidare på IP och möjliggör en förbindelse mellan två värddatorer som kan utbyta data. Protokollet kontrollerar att alla data skickas innan överföringen avslutas. TCP kopplas ofta samman med IP för att skapa en TCP/IP-anslutning.

Vid överföring ansluts parterna med en trevägs handskakning. Det innebär att klienten först skickar ett datasegment kallat SYN (synchronize) till värddatorn. Värddatorn godkänner synkroniseringen med SYN-ACK-segment (synchronize-acknowledgement). Därefter skickar klienten ännu ett SYN-segment för att etablera anslutningen.

Med anslutningen etablerad kan data skickas, och mottagaren skickar ett ACK-segment för varje mottaget paket. Vid utebliven ACK-signal från mottagaren skickar sändaren om paketet. På så sätt säkerställs att inga data går förlorat. När överföringen är klar skickas ett avslutningssegment kallat FIN (finish). FIN-segmentet besvaras med ett ACK-segment och därefter stängs anslutningen ned. [11]

2.1.3 DNS

På svenska domännamnssystem. Kopplar ihop domännamn med IP-adresser. Ett steg som förenklar nätverkskommunikation, då domäner är behändiga att jobba med, jämfört mot IP-adresser, då de kan ges intuitiva namn.[12]

2.1.4 Socket

Typ av virtuell kontakt, vilken dataöverföring kan ske igenom. Använder sig vanligtvis av IP vid överföring, tillsammans med en port. Adressen och porten anges då enligt IP:port, exempelvis *127.0.0.0:3333*. [13]

2.2 Positionering

Bilen behöver uppgifter om sin position i förhållande till infrastrukturen för att kunna leverera korrekt data för analyserna. Exempelvis behövs denna för att beräkna vilken hastighet som skall hållas för att nå trafikljuset vid körbar status, eller för att hitta närmaste infrastruktur i sin omgivning. Den aktuella bilmodellen saknade system för positionering.

3. Teknisk bakgrund

3.1 Raspberry Pi

Raspberry Pi är en enkortsdator ungefär lika stor som ett kreditkort med ett micro-SD-kort som hårddisk. Man installerar ett operativsystem på microSD-kortet via en dator och ansluter det sedan Raspberry Pi. GPIO-portarna på Raspberryn möjliggör installation av tillbehör som olika displayer, dioder, motorer, olika typer av sensorer, mottagare och sändare.[14] I detta projekt har man använt Raspbian som operativsystem för infrastrukturen.

3.2 Python

Python är ett programspråk som stödjer flera programmeringsparadigmer; objektorienterad, imperativ, funktionell och procedurell.
Python använder en programtolk istället för en kompilator. Detta ger generellt sett långsammare exekvering av program, då en kompilator genererar färdig maskinkod som senare kan exekveras direkt på processornivå. Dock är många Python-bibliotek, så pass optimerade att de körs i när nog samma hastighet som kompilerande språk. [15]

3.3 ZeroMQ

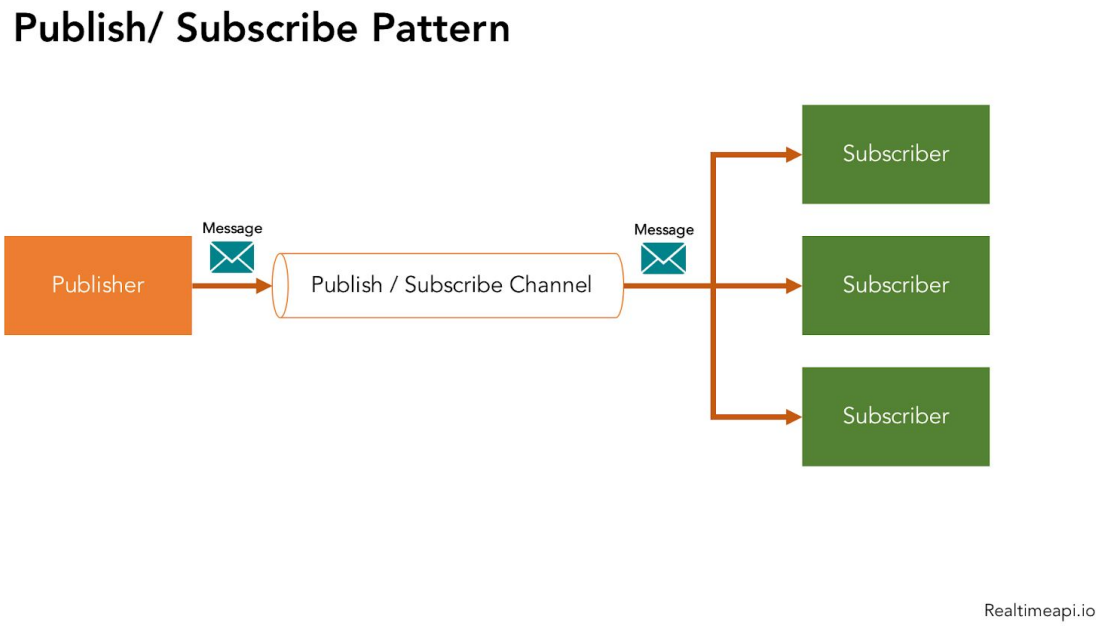
ZeroMQ (ØMQ) är ett programmeringsbibliotek som används för att skapa sockets. ZeroMQ är skrivet i C++ men portat till många språk, däribland Python.
ZeroMQ baserar sina sockets på kommunikation med TCP. När en socket skapats kan en klient ansluta till den för att skicka och ta emot data. ZeroMQ är kapabel att överföra data endast som bytes. ZeroMQ har gränssnitt för många språk, däribland Python. ZeroMQ använder olika sockets för olika strukturer av dataöverföring. [16]

3.3.1 PUB/SUB

ZeroMQ har en överföringsstruktur som kallas **PUB-SUB**. Det består av två sockets, en publisher **PUB** och en subscriber **SUB**.
PUB är en utgivare som skickar ut meddelanden. SUB är en abonnent, som kan prenumerera till en **PUB** för att ta emot meddelanden.

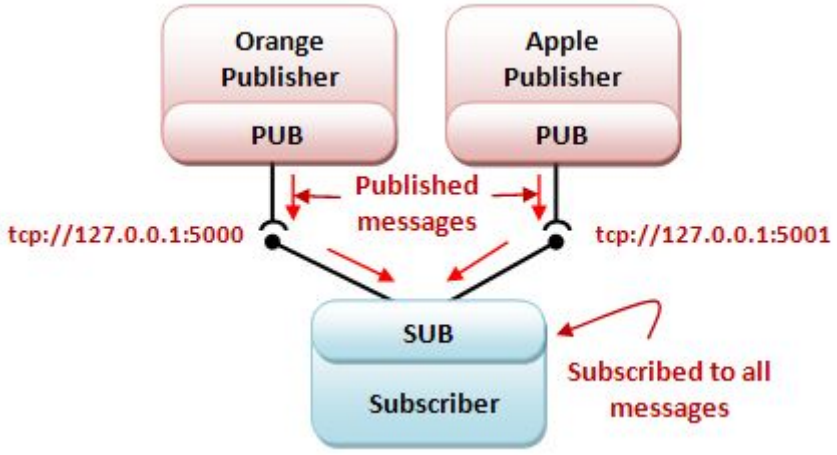
PUB för ingen statistik på antalet prenumeranter.
En **SUB** kan även prenumerera på datan från flera PUB. [17]

Ett exempel på fall som används mer ofta är att flera abonnenter så kallade SUB prenumererar på meddelanden som publiceras av utgivaren PUB som illustreras i figur 3.1



Figur 3.1: Illustration av PUB-SUB socket-kommunikation.[18]

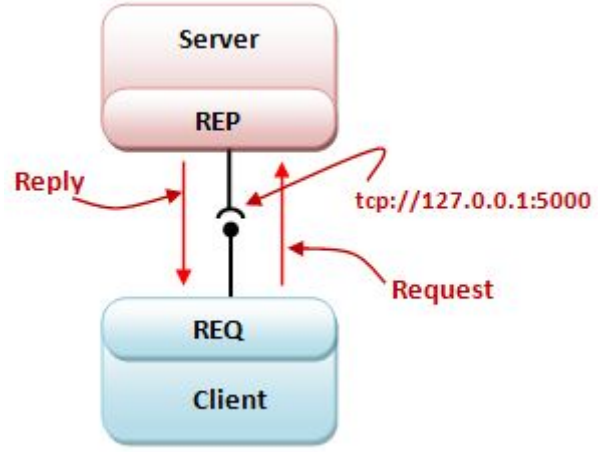
En SUB kan även prenumerera på meddelanden från flera PUB. Ett exempel för SUB med flera PUB visar i Figur 3.2 som visar en SUB som prenumererar på 2 olika PUB, en med namn Orange PUB och en annan med namn Apple PUB.



Figur 3.2: Illustration av PUB-SUB socket-kommunikation.[19]

3.3.2 REQ-REP

Den enklaste strukturen för att skicka information via ZeroMQ är med en socket av typen require, REQ, och en socket av typen reply, REP.
REQ agerar som en klient och REP som en server. Till skillnad från PUB/SUB har REQ/REP alltid en specifik avsändare och mottagare. REQ/REP förväntar sig alltid ett svar genom att ej skicka nya paket tills REP kvitterat mottagandet. [20]



Figur 3.3: Illustration av REQ--REP socket-kommunikation.[21]

3.4 Protocol buffers

Protocol buffer,kort protobuf är en teknik utvecklad av Google för att översätta strukturerad data till ett format lämpat för överföring. Den översatta datan överförs, och rekonstrueras med hjälp av protobuf till strukturerad data igen.

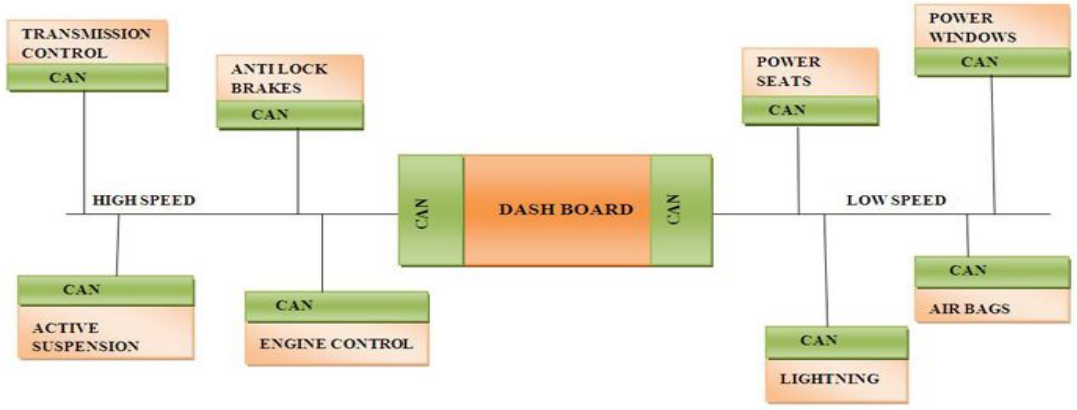
I detta projekt används Protobuf tillsammans med ZeroMQ, eftersom ZeroMQ endast överför bytes. Datan som skall skickas är vanligtvis heltal eller textdata, formaterade som *int* respektive *string*, och därför måste den översättas till bytes för att kunna hanteras av ZeroMQ. Tack vare protobuf behöver inte datan avkodas manuellt, och en stor fördel med det är att meddelande som skickas inte måste ha en specifik plats i byten, platserna hanteras av protobuf. [22]

3.5 Positionering

Enligt kravspecifikationen skulle utvecklingen ske med en bil som hade ett fungerande system för positionering.Tanken var att plattformen skulle vara utrustad med ett system för självkörning, där positionering skulle ingå. Dock hade detta ej hunnits implementeras av Infotiv innan vi påbörjade examensarbetet. Därför mäts position baserat på hastigheten under ett specifikt tidsintervall.

3.6 CAN-buss

CAN, kort för Controller Area Network är en kommunikationsmetod mellan olika elektronikenheter inbyggda i ett fordon. CAN-buss är en seriell databuss med kommunikation enligt CAN-protokollet. CAN-buss kommunikation skapades av Bosch 1983 för användning i fordonsindustrin.
~~Det~~ Protokollet gör det möjligt att flera styrenheter i bilen kan skicka meddelanden och flera styrenheter kan motta meddelanden simultant, snabbt och säkert. [23] Se figur 3.3.

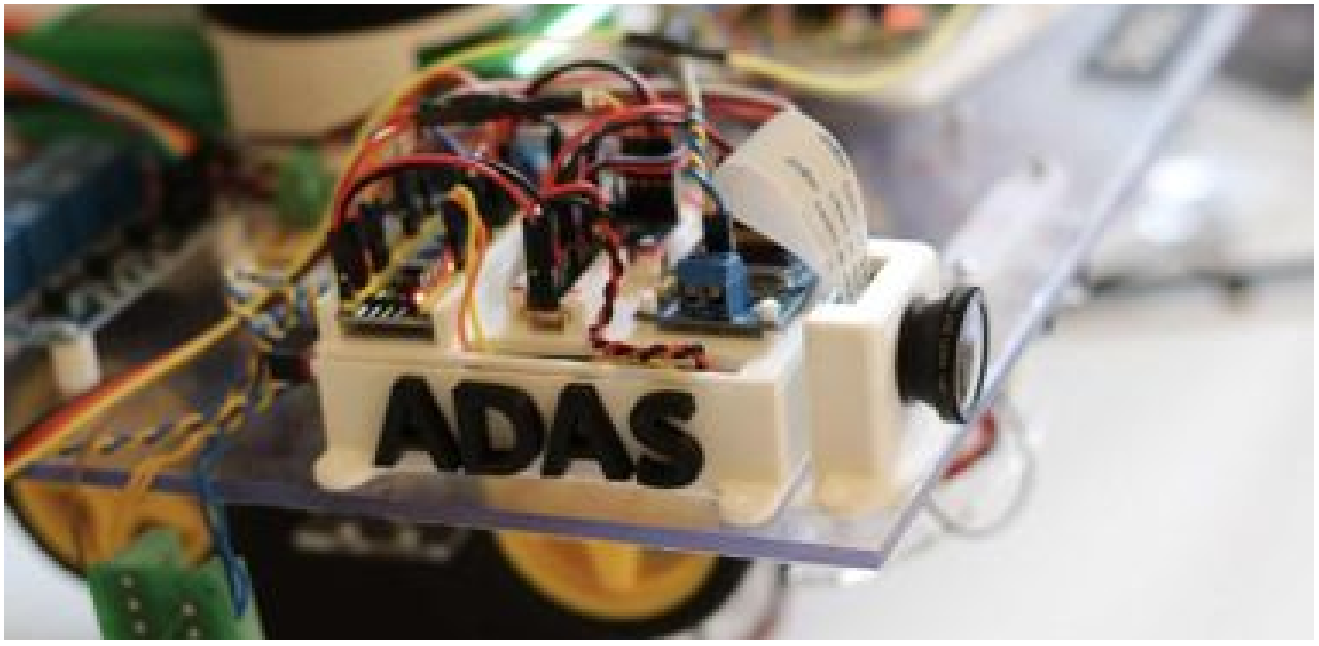


Figur 3.4: Visar ett exempel på CAN-kommunikation mellan olika styrenheter i ett fordon.[24]

3.7 ADAS

ADAS är en akronym för Advanced Driver Assistance System som används för enbart motorn. ADAS används för att validera alla signaler på bilen innan de går till motorn. Enheten består av en mikroprocessor, en Raspberry Pi och en CAN-modul.

ADAS prioriterar alla signaler som de olika enheterna på bilen vill sända till motorn. Om ett system till exempel upptäcker en kollisionsrisk så är det viktigare att ta hänsyn till än en signal som säger att trafikljuset är grönt. ADAS mikroprocessor kan ta beslut i realtid utifrån all denna data. [25]



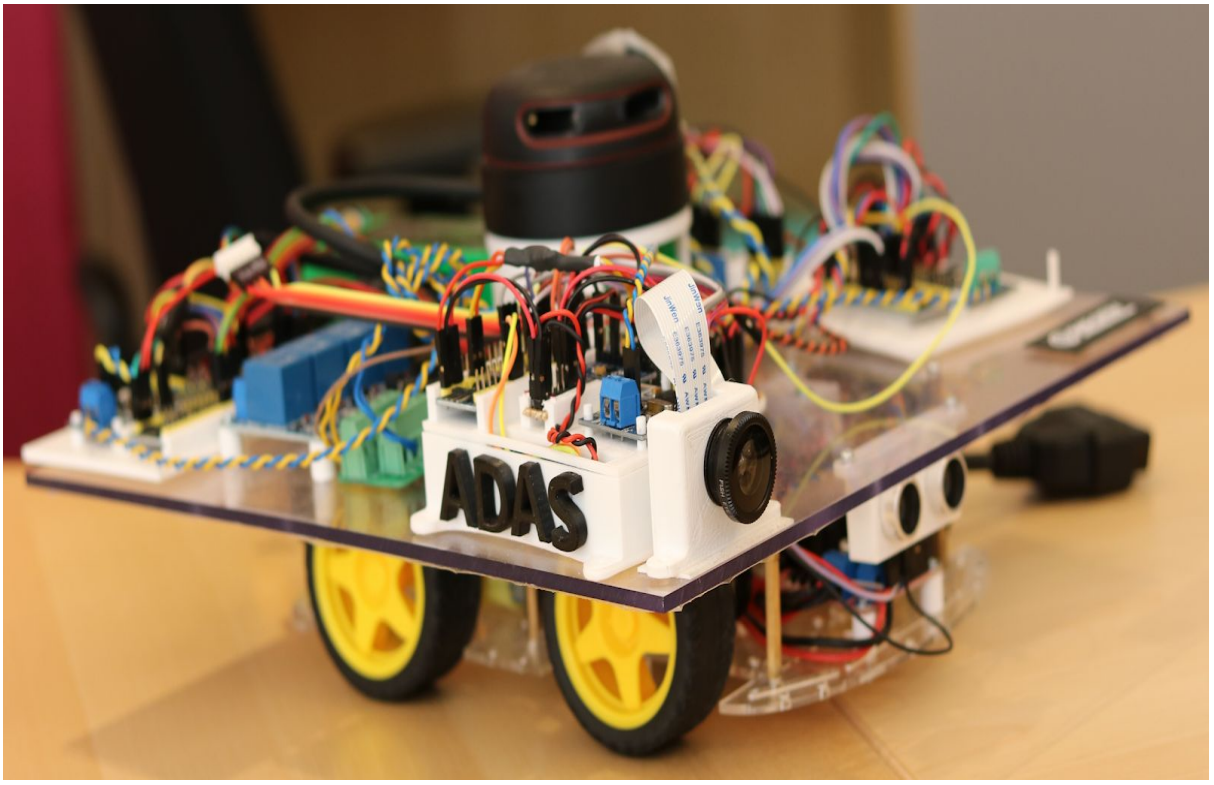
Figur 3.5: Illustration av ADAS. Egen bild, tagen av bilen på Infotiv.

3.8 Infotiv Autonoma Plattform

I detta stycke presenteras Infotivs autonoma bilplattform kort.
Infotiv Autonom Plattform är en styrsystemsplattform, byggt som en modell av ett fordon. I fortsättningen benämner vi den “bilen”. Bilen använder fyra hjul för att navigera runt i sin omgivning. Varje hjul har en elmotor som är monterat på den undre plattan.

Plattan har dimensionerna 30 x 30 cm och består av olika sensorer, en Raspberry Pi samt CAN-bussar. I figur 3.3 syns i det högra hörnet ADAS tillsammans med en kameralins. De är kopplade till Raspberry Pi-datorn på den övre plattan. Bakom Raspberryn kan en LIDAR-sensor ses. På bilen finns elektriska styrenheter, Lidar, ultraljud och en kamera. De olika enheterna är uppdelade på ett sätt som efterliknar det elektriska systemet i en modern Volvo-bil. Sensor data från de olika sensorerna publiceras på bilens lokala nätverk på Raspberry Pi-n via socket-kommunikation. Socket-kommunikationen sker via Python-bibliotek.

Det gör det möjligt att ansluta en uppsättning av publisher till en uppsättning av subscriber sockets för att hantera nätverkskommunikation i vårt projekt mellan infrastrukturen och bilen. Signaler eller instruktioner kan också skickas via dessa sockets för att bestämma exempelvis hjulhastighet eller svänghastighet.



Figur 3.6: Illustration av Infotiv Autonoma Platform. Egen bild tagen av bilen på Infotiv.

4. Kravspecifikation

Kraven för detta arbete är att utvärdera möjligheterna med C2I- kommunikation för autonoma beslut av fordonet och implementera detta på bilen. Bilen skall kunna kommunicera med infrastruktur och en annan bil för att ta följande beslut:

A) Ljussignalbeslut

Bilen skall redan innan den kommer till en ljussignal kunna ta emot signalens status samt kommunicera med en annan bil för att avgöra om farten bör sänkas för att synkronisera med grönt ljus, eller om man istället skall ta en omväg för att minska försening.

B) Brobeslut

Bilen skall redan innan den kommer till bron kunna ta emot bro-status från bron för att avgöra om farten bör minskas eller ökas eller om man istället skall ta en omväg för att minska försening.

C) Beslut i vägkorsning utan trafikljus

Detta skall göras genom att två bilar skall kunna kommunicera inte bara mellan sig själva utan även med infrastrukturen. Bilarna skall komma överens om vem som skall ha företräde. Om bilarna har lika stort avstånd skall de komma överens om prioritering.
Som redundans skall en funktion som stoppar bilen byggas in om kommunikationen felar.

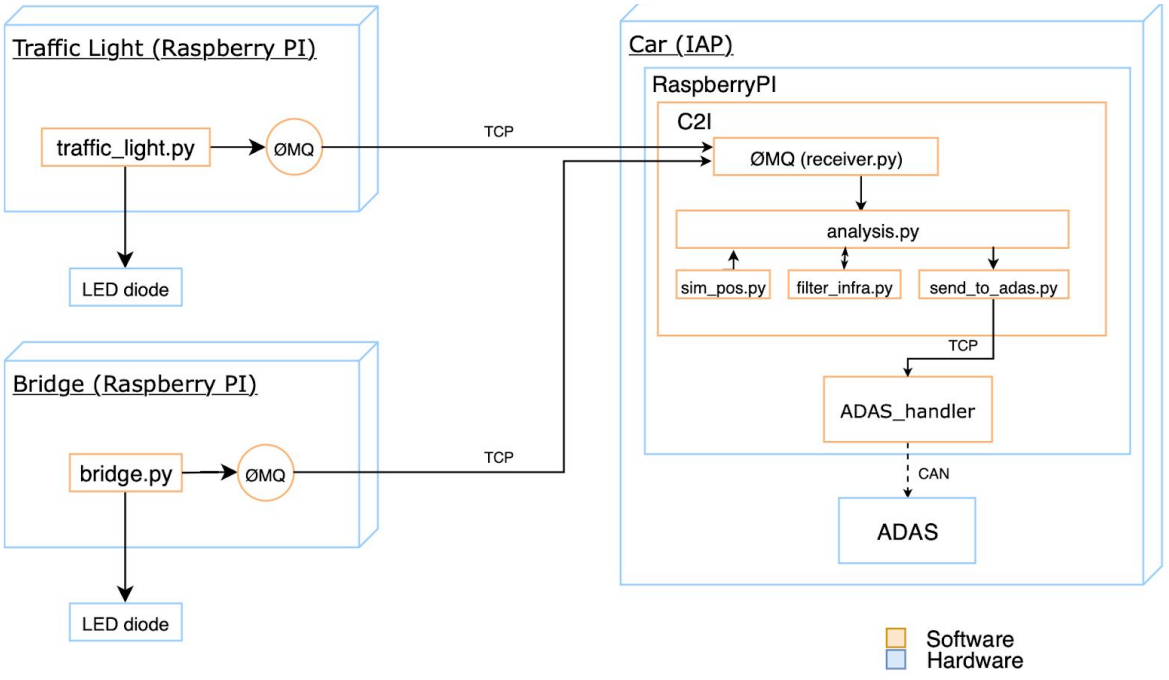
5. Instudering och systemarkitektur

Som ett första steg utfördes efterforskningar av tidigare tillämpningar där kommunikation mellan trafik och infrastruktur använts. Exempelvis fanns koncept med smarta vägar där krockar kan förebyggas, men även enklare tillämpningar där smarta parkeringsautomater kommunicerar med fordon. Tillämpningarna diskuterades under en brainstorming, där även nya idéer diskuterades, samt vilka som var lämpligast för projektet. Under brainstormingen ansågs Infotivs önskemål med trafikljus och bro vara rimliga.

Med en bild om vilken teknik som skulle utvecklas behövdes mer kunskap inom kommunikation. För att snabbt komma igång studerades modulen *C2C* på bilen, vilken hanterar kommunikationen mellan bilarna. Modulen är skriven i Python och använder särskilda bibliotek.

En gemensam slutsats drogs att vid utvecklingen av C2I måste först och främst en infrastruktur för kommunikationen tas fram.
För att kunna överföra datan måste serialisering och deserialisering av datan specificeras.
Till slut måste fordonet ta intelligenta beslut utifrån den överförda datan.

5.1 Systemarkitektur



Figur 5.1 Systemarkitektur av C2I-systemet.

Systemarkitekturen i figur 5.1 illustrerar systemet. Till vänster i figuren syns infrastrukturerna, *Traffic Light* samt *Bridge*, som består av Raspberry Pi. De kommunicerar med bilen genom ZeroMQ (stiliserat Ø i figuren) över TCP. LED-dioder är även kopplade till Raspberry PI:s GPIO.

På bilen (CAR) syns (Car to Infrastructure) C2I:s struktur:

1. En ZeroMQ-klass kallad *receiver* tar emot data från infrastrukturen som läses av analysklassen *analysis*.

2. Analysklassen tar även emot data från simulatorn för positionering som skickas vidare till filtreringsklassen.

3. Filtreringsklassen returnerar vilken infrastruktur som ligger närmast bilen. Detta är tillräckligt för att analysklassen skall kunna beräkna lämplig hastighet och planera en färdväg som den skickar till ADAS genom *send_to_adas*.

4. Send_to_adas serialiserar datan och skickar den via ZeroMQ REQ-REP-socket till en ADAS-hanterare som finns på bilen (Car).

5. ADAS-hanteraren skickar slutligen meddelandet via CAN till ADAS-modulen på bilen (Car).

5.2 Utvecklingsmiljö

C2I är utvecklat i Python. För textredigering användes den integrerade utvecklingsmiljön *Visual Studio Code*. Programvaran för C2I är beroende av *ZeroMQ*, *Protobuf* och *Google*.

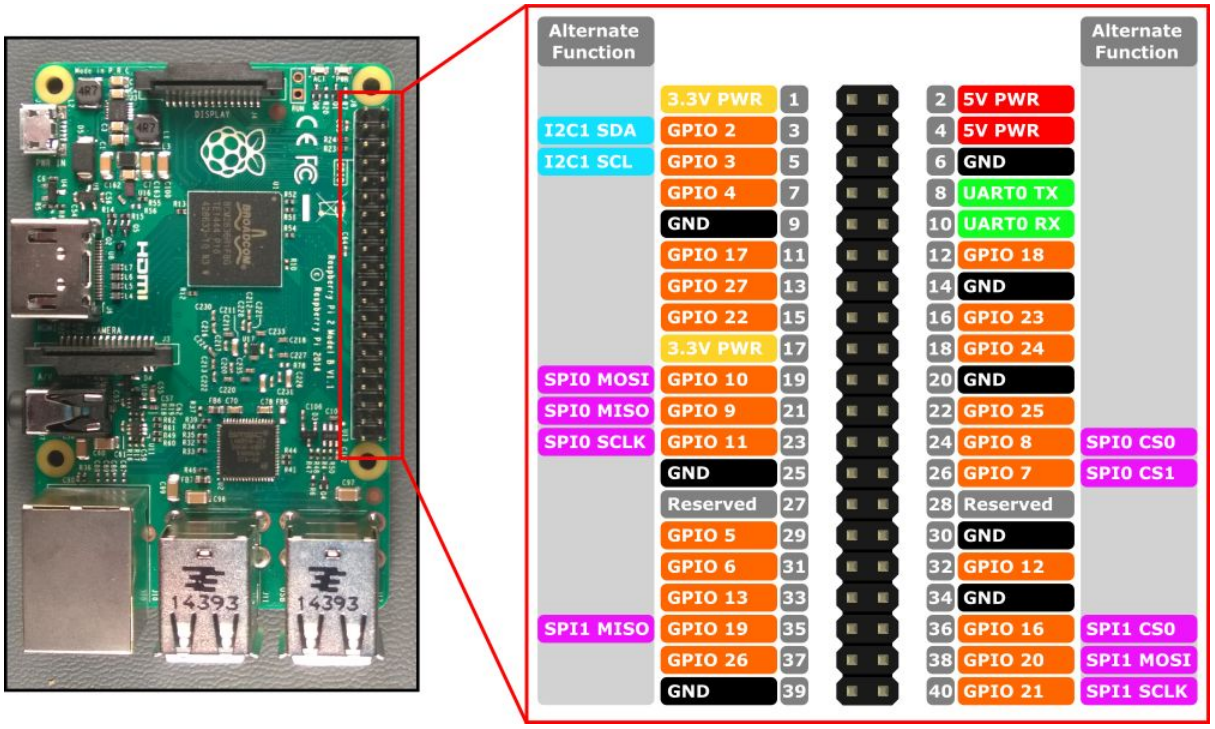
För att simulera infrastrukturer utanför värddatorn används Raspberry Pi med en installation av NOOBS OS. [26]
För att kunna kommunicera krävs Python med samma bibliotek, det vill säga *ZeroMQ* med *Protobuf* av *Google*.

Bilen använder en Raspberry Pi för att köra C2I.

6. Plattform för hårdvara

Det fanns inget dedikerat trafikljus eller bro för att kommunicera med. Detta ledde till ett behov av ta fram hårdvara för infrastrukturen i form av ett trafikljus och en bro.

En Raspberry Pi Model 3 B användes för trafikljuset och en annan Raspberry Pi Model 3 B för bron som visas i Figur 6.1

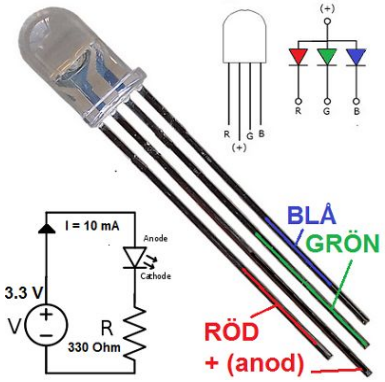


Figur 6.1: Raspberry Pi med input/output pinnar tabell. [27]

6.1 Hårdvara för infrastrukturen

RGB LED diod är en kombination av 3 lysdioder: 1 styck röd LED, 1 styck grön LED, 1 styck blå LED. Färgen som bildas av RGB-LED är en kombination av färgerna på var och en av dessa tre lysdioder. Det finns 2 sorters RGB-LED dioder, en med gemensam anod RGB-LED eller med gemensam katod RGB-LED. RGB-LED-dioder har 4 ben som är olika långa. Den längsta är jord (-) eller spänningen (+) beroende på om den är en gemensam katod eller den gemensamma anod-benet. De andra tre benen motsvarar rött, grönt och blått [28]. Se figur 6.2 nedan.

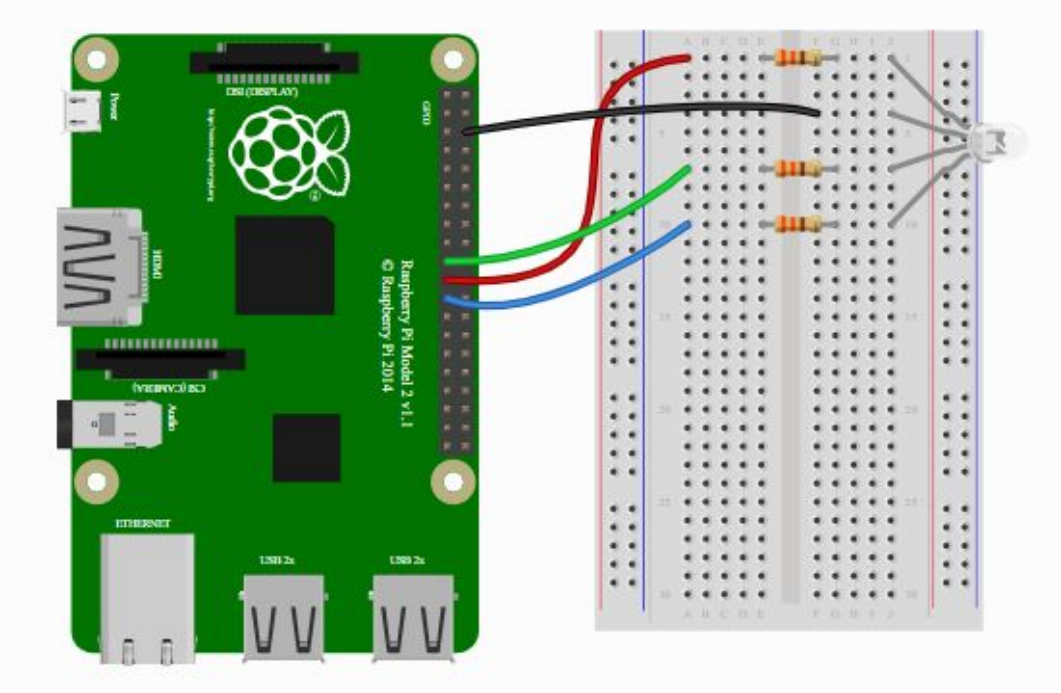
.



Figur 6.2: En RGB-LED diod med olika ben och möjliga färger som man kan visa.[29]

Två stycken RGB LED-dioder med gemensam anod (+) kopplades på ett kopplingsplatta som i sin tur kopplades till en Raspberry Pi.

RGB-LED-dioden anslutes via motstånd till en varsin IO-pinne på kretskortet till höger om Raspberry Pi för att på så sätt styra önskad färg. RGB-LED dioderna tillsammans med resistorerna löddes fast på ett kretskort som sedan monterades på trafikljuset och bron. Kretskorten kopplades i sin tur till Raspberry Pi-enheten. Figur 6.3 visar kopplingen i prototypstatus med kopplingsdäck.



Figur 6.3: Visar kretskopplingen för en RGB-LED diod på en kopplingsplatta som är vidarekopplat till en Raspberry PI.[30]

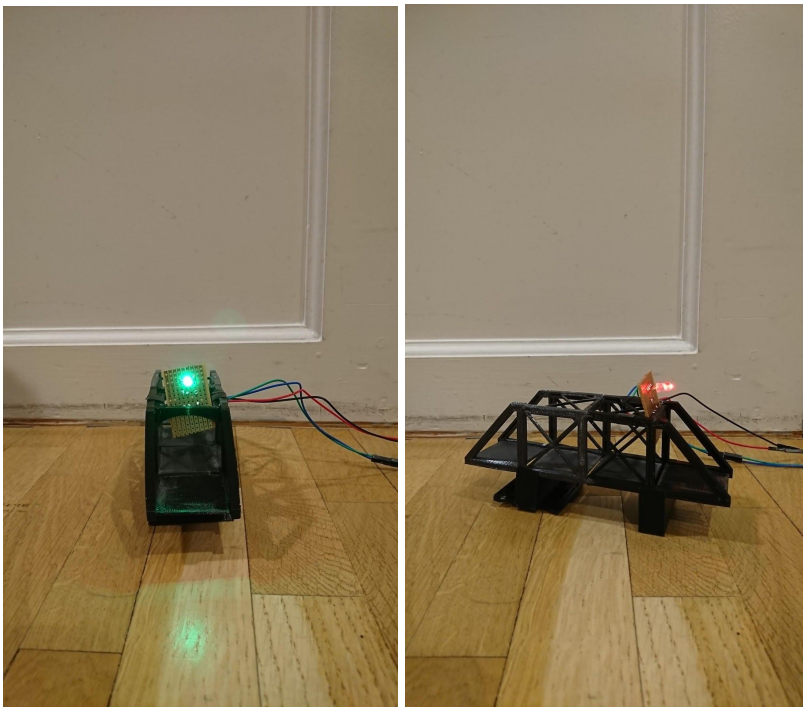
GPIO-signalerna initierades och styrdes sedan i programmet för trafikljuset och bron, ett exempel hur detta gjordes visas i figuren nedan. Figuren visar hur man initierar två stycken GPIO pinne nummer 23 och port nummer 24 på Raspberry Pi till utgångar.

```
def init_diode(self):
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(23, GPIO.OUT)
    GPIO.setup(24, GPIO.OUT)
    self.set_diode(self.status)
```

Se **Appendix** för mer information om numrering av input och output pinnar på Raspberry Pi datorn.

Den färdiga versionen av trafikljuset och bron. Stativ och lådor fanns tillgängliga. Se figur 6.4 och 6.5 nedan.

Figur 6.4 Illustrationer av trafikljuset i två olika ljussignaler/färger rött och grönt. Egna bilder tagna på Infotiv.



Figur 6.5: Illustrationer av bron i två olika ljussignaler/färger, rött och grönt. Egna bilder tagna på Infotiv.

7. Mjukvara

7.1 Processer i trådar och resurshantering

C2I exekveras i trådar för att flera operationer skall köras simultant. Trådad programmering implementeras i systemet med hjälp av *threading*-biblioteket som levereras med Python.

När ett program utnyttjar flera trådar kan det ibland ske en krock om flera trådar använder samma resurs samtidigt. För att skydda programmet från oförutsägbart beteende låser man resurser då en tråd använder de, även kallat ömsesidig uteslutning. Låsningen går till så att tråden som först hämtar låset får tillgång till resursen. Om en tråd försöker använda en upptagen resurs måste den alltså vänta tills den första tråden släpper låset. Den andra tråden sätts i en kö till resursen och blir idle fram tills dess. Låsningen garanterar säker egenåtkomst. [31]

7.2 Mjukvara för infrastruktur

Infrastrukturernas program för trafikljus och bro är väldigt lika. Både bron och trafikljuset instansierar ett objekt vid uppstart. Objektet läser infrastrukturens status, d.v.s. öppen/stängd för bron eller färgen för trafikljuset. Bron beräknar även framtida status färg för trafikljuset eller bron för att kunna ge bilen information om att en alternativ rutt går snabbare än att vänta på brostängning.

Trafikljuset består av en klass med fem klassmetoder, nedan följer en beskrivning av dessa metoder.

update_values
Uppdaterar infrastrukturens värden för bland annat status, tid till statusbyte och nästkommande status.

Metoden räknar nedåt från en angiven tid till 0. När 0 nås anropas next_status för att uppdatera trafikljusets värden som är färg, därefter uppdateras tidsstämpeln. Som används för att kontrollera om meddelandet är unikt på mottagarsidan.

next_status
För att uppdatera statusvärden anropas next_status av update_values. Vid anrop beräknas vilken status som kommer näst i tur, varpå status samt last_status uppdateras. next_status uppdaterar även trafikljuset diod till rätt färg.

traffic_to_bytes
En metod för att serialisera data inför publicering med ZMQ. Initierar ett protobufobjekt vilken trafikdata sparas ned i för att returneras serialiserad data.

init_diode
Initierar en diod som är kopplad till trafikljusets Raspberry Pi. Använder ett Raspberry Pi-bibliotek, GPIO (akronym av *general purpose input output*), för att ställa in två stycken pinnar på raspberryn till utgångar.

set_diode
Har en inparameter för vilken status som är aktiv. Först nollställs pinnarna, så dioden hamnar i neutralt läge, därefter sätts rätt pinne eller pinnar till hög nivå för att ändra färg på dioden.

När klassen är definierad instansieras ett objekt som representerar trafikljuset i kod. En PUB-socket skapas med hjälp av ZeroMQ. Varje sekund publiceras serialiserad data till socketen med information om status, föregående status, tid tills statusbyte, samt en tidsstämpel.

7.3 Simulering av position

I avsnitt 3.6 nämndes att bilen ej var självgående under exjobbet, vilket stod som en förutsättning i kravspecifikationen. Istället infördes en klass tidigt i projektet som beräknar en ungefärlig position beroende på bilens hastighet per tidsintervall om 1/10 sekund. Positionen beräknas bara i ett led. Detta på grund av att bilens signaler för att svänga får bilen att svänga olika mycket varje gång, med precis samma begynnelsevillkor. Tekniken förenklades därför till att använda position i en dimension för att allokera tid till andra delar.

Hur fungerar då hela modulen? Precis som infrastrukturerna består av en klass gör även bilens moduler detta. Positionsmodulen har en konstruktor som tar emot ett bil-objekt för att kunna läsa bilens hastighet. Denna hastighet används sedan i en metod ***car_pos_thread***. Så länge som hastighetssignalen till motorn är tillräckligt stor för att bilen skall röra på sig uppdateras en klassvariabel ***car_pos***. ***car_pos*** nya värde blir dess tidigare värde plus bilens hastighet per det tidsintervall som räknas med multiplicerat med tidsintervallet. Med andra ord adderas sträckan under tidsintervallet till bilens totala sträcka.

Som namnet antyder körs car_pos_thread i en tråd för att alltid uppdatera positionen när programmet är igång, vilken skapas när klassen instansieras. Ett lås används därför för att öka tillförlitligheten vid åtkomst till variabeln ***car_pos***. En metod *get* förenklar hämtning av positionsdatan genom att automatiskt begära ett lås innan hämtning.

7.4 Filtrering av infrastrukturer

När positionen är inhämtad kan programmet beräkna vilken infrastruktur som är närmast och inte passerat redan, för att sedan ansluta till den. En klass utför detta med två metoder, en metod för att hämta infrastrukturer och en för att välja ut rätt infrastruktur.

Alla omkringliggande infrastrukturer ligger i en Python-ordlista. Metoden ***retreive_infrastructures*** läser vilken position dessa infrastrukturer har, för att välja ut de som ligger inom en avgränsning på 800 meter. De som är tillräckligt nära sparas i en ny lista.

Den nya listan läser metoden *select* in. Genom att iterera över alla objekt i listan och jämföra deras positioner med varandra väljs sedan den infrastruktur ut som är närmast. När en ny infrastruktur är närmre än föregående uppdateras klassvariablerna för IP och port till den nya infrastrukturens. Metoden flaggar då för förändring och en förbindelse upprättas med den nya infrastrukturen.
Båda dessa metoder körs i trådar för att uppdatera datan i bakgrunden. Det finns många hjälpmetoder för att hämta data med lås. Metoden för att hämta flaggan för förändring återställer även flaggan till *False*.

7.5 Kommunikation av bil till infrastruktur

Bilen kommunicerar med trafikljuset och bron genom att skapa förbindelser med infrastrukturen. För att åstadkomma detta krävs det en klass som hanterar kommunikationen. All kommunikation sker genom en instans av klassen *receiver*. När klassen instansieras skapas direkt en förbindelse till den infrastruktur som filtreringsklassen valt ut. Detta genom att skapa en ZMQ subscriber socket. Socketen hämtar data som publiceras av infrastrukturen som sedan sparas ned i klassvariabler.

Socketsarna skickar datan trådlöst över wi-fi. Som nämnt i avsnitt 3.4 skickar ZeroMQ endast bytes över nätverket. Därför implementeras även protobuf i receiverklassen i klassmetoden *update_values*.

Metoden jobbar i en tråd med att uppdatera datan ett visst antal gånger per sekund, där varje uppdateringen utför tre steg:

1. *Ett protobufobjekt instansieras*
2. *Data hämtas och avserialiseras*
3. *Datan sparas i variabler*

Steg 1 instansierar ett objekt från protobuf så datan skall kunna avserialiseras i steg 2. Datan i steg 2 hämtas från socketen med hjälp av ZeroMQs pollersocket. Pollersocketen gör 1000 avläsningar på subscribersocketen och sparar dessa i en Python-ordlista.

Metoden kan sedan jämföra dessa avläsningar för att se om data har publicerats på socketen. Detta är fördelaktigt för då kan en läsning göras på socketen utan att blockera programmet tills en signal tas emot, då man redan vet att en signal är tillgänglig på socketen.
Datan avserialiseras i samband med läsningen. Återfinns inte någon information på socketen sätts en felflagga som låter analysklassen veta att infrastrukturen inte har skickat någon data under föregående iteration.
I steg 3 sparas datan från socketen ned i rätt variabel.

7.6 Analys av signaler

Klassen *analys* instansierar övriga objekt. Analys hämtar sedan data från de övriga objekten. Datan består bland annat av infrastrukturernas status och position eller information om bilens hastighet. Analys kopplar ihop datan och tar sedan ett beslut om hur bilen skall uppföra sig i förhållande till närliggande infrastruktur.

Själva metoden som utför analysen utför liknande steg oberoende av vilken typ av infrastruktur den jobbar mot. Detta möjliggör generella steg som att beräkna avstånd till den infrastruktur som är näst på tur att passeras och ansluta till den. Därefter begär programmet information om vilken typ av infrastruktur det är för att lämplig metod för infrastrukturen.

Analysen för trafikljus till exempel tar emot vilket tillstånd trafikljuset kommer att vara i då bilen passerar. Om trafikljuset beräknas vara grönt (eller gult med nästa status grönt) körs bilen enligt högsta tillåtna hastighetsbegränsning. Vid andra trafiksignaler beräknar programmet den hastighet som ger föraren minsta möjliga väntetid vid trafikljuset. Om det är möjligt att anlända vid trafikljuset utan att bromsa kommer algoritmen justera till denna hastighet.

7.7 Detektering av opålitliga signaler

Datan som bilen läser från infrastrukturen förutsätts även tas emot av andra bilar eller användare i andra bilar. Denna information om hur andra bilar kommer uppföra sig används sedan för att påverka bilens hastighet i ett mikroperspektiv, och trafikflödet i ett makroperspektiv. Ur detta hänseende ställs ett krav på att datan är pålitlig.

I programmet testas därför signalerna genom en jämförelse mot föregående signaler. Varje signal som skickas från infrastrukturen har en tidsstämpel. Tidsstämplarna sparas i en lista med plats för ett specificerat antal element. Det äldsta elementet skrivs kontinuerligt över av det senaste vid varje hämtning av ny data. Skulle hela listan innehålla samma tidsstämpel antas att infrastrukturens logik eller kommunikation är trasig och därmed går informationen ej att lita på. I nuvarande implementation hanteras detta med att bilen stannar.

7.8 Skicka signaler till motor via ADAS

Bilen har en implementation för att prioritera signaler, ADAS, som nämnts i kapitel 3.6.1. För att skicka datan används en klass med sju metoder. En metod används för att placera signalerna i en kö, en används för att kommunicera med ADAS och resterande fem används för att skapa en signal för ett visst beteende, till exempel *starta motorn*.
Metoden som placerar signaler i en kö använder sig av en Python-kötyp *PriorityQueue*. Metoden har en inparameter avsedd för motorsignalen, vilken placeras på sista platsen i kön.

Metoden för att kommunicera med ADAS läser det äldsta meddelandet från kön och börjar med att kontrollera att det är inom tidsramen för att betraktas som aktuellt. Uppfyller det tidskravet och ej är tomt skickas det via en lokal socket till ADAS.

De övriga fem metoderna genererar en signal som motorn kan läsa. På bilen finns ett bibliotek som gör om data, så som hastighet, till signaler som elmotorerna på bilen kan läsa. Protobuf hade varit ett smidigare val även i denna klassen, men gick ej att realisera eftersom ADAS ej implementerat stöd för protobuf ännu. Med signal som motorn kan läsa genererad, sparas denna sedan ned i kön med hjälp av metoden beskriven ovan.

8. Resultat

Syftet med examensarbetet var att få ett autonomt fordon att kommunicera med en omgivande infrastruktur för att på så sätt kunna ta beslut på liknande sätt som människor. Man kan konstatera att detta har uppnåtts.
Bilen är i dagsläget kapabel att kommunicera med både trafikljus och broar.

Målet var att realisera ett system för en modell av kommunikation mellan ett autonomt fordon och infrastruktur. Detta innefattade att utveckla eventuell hårdvara samt att ta fram mjukvara som möjliggör att infrastrukturen skulle publicera data via ett nätverk. Infotivs bil skulle kunna ta emot och agera enligt informationen i den sända datan. Bilen skulle även kunna räkna ut vilken infrastruktur som kommer passeras härnäst och agera utifrån detta.

Bilen klarar dessutom av att söka upp närliggande infrastrukturer och filtrera bort de som redan passerats. Därefter tar bilen emot information från den enhet som är näst på tur att passeras. Observera att detta inte var något krav från företaget, men det underlättade andra delar av implementationen samtidigt som det förbättrade demon. Beroende på hur nära bilen är till infrastrukturen och vilken status infrastrukturen har, skall bilen agera enligt informationen. I dagsläget kan bilen ta beslut enligt följande användarfall:

- Trafikljuset skickar signal status röd om t.ex 2 sekunder, då bromsar bilen och bilen kommer till full stopp.
- Trafikljuset skickar signal status gul om t.ex. 8 sekunder, då tar bilen en alternativ rutt om det finns möjlighet.
- Trafikljuset skickar signal status grönt om t.ex 3 sekunder, då fortsätter bilen köra.

Bilen kan dessutom hantera felsignaler och inaktuella signaler från infrastrukturen som är en extra funktion som togs fram under utvecklingen.
Företaget Infotiv är väldigt nöjda med resultatet. Mjukvaran beter sig bra och deras önskemål möttes upp till all förväntan.

Vi har utvecklat en mjukvara som hanterar kommunikation mellan fordon och infrastruktur som uppfyller målen för projektet.

9. Diskussion

Det som specificeras i avgränsningarna var att bilen skulle endast kommunicera med två typer av infrastruktur, trafikljus och broar med broöppning.
Det tredje kravet d.v.s att etablera autonoma implementationer av bilen som klarar av även korsningar utan trafikljus implementerades inte på grund av avsaknad av tillämpning av detta eftersom bilarna saknar riktig positionering.
Detta skulle påverka säkerheten. Det skulle bli för hög osäkerhet när båda bilarna simulerar en position. Det i sin tur skulle göra det praktiskt omöjligt för bilen att räkna ut var den andra bilen är eftersom bilarnas simulerade positioner ej är förankrade i rummet.

Gällande kommunikationen, i verkligheten skulle det ske med ett DSRC-protokoll (dedicated short range communications) utvecklat för bilar på frekvenser mellan 5,8 till 5,9 GHz. Då tillgång till DSRC ej fanns används wi-fi istället. Wifi-nätverket hade en liknande frekvens på 5.0 GHz, och därför uppnås en tillräckligt låg latens.

Vi spenderade mycket tid på att hitta en tillfredsställande lösning till att räkna ut bilens position. SLAM-algoritmer testades, och fungerade med hjälp av en version av ROS på bilens Raspberry Pi, men den här lösningen var väldigt tidskrävande och släpptes till slut för att hinna med andra mål. Hade denna tiden utnyttjats till att förfina bilens nuvarande positioneringssystem, skulle slutresultatet antagligen blivit bättre.

Ytterligare forskning kan göras på följande områden:

1. Att ge bilen en realtidsposition eller att bilen kan lokalisera sig på ett visst rutnät eller en väg. Då kan ytterligare forskning göras om att få 2 autonoma bilar att fatta beslut och korsa korsningar samtidigt utan signaler.

2. Det kan också vara intressant att undersöka autonoma bilar beteende på motorvägen, exempelvis att autonoma bilar kan kommunicera med infrastrukturen på motorvägen, samtidigt som de kommunicerar med varandra, informera bilar närmast av till exempel byte av fil, problem som har uppstått på vägen för att förbättra säkerheten och trafikflödeseffektiviteten.

3. Vidareutveckling och Implementering av kommunikation mellan kameran på bilen och infrastruktur. Detta kan leda till fortsatt utveckling av både bil till infrastruktur kommunikation och machine learning.

9.1 Miljöaspekter

Det är inte lätt att förutse hur autonom teknik kommer påverka miljön förrän autonoma elbilar får en större spridning. Studier och fakta visar att det finns både positiva och negativa sätt autonoma elbilar kommer att påverka miljön.

Autonoma bilar är oftast laddbara elbilar som använder litium-jon batterier eller liknande för framdrivning. Autonoma fordon är programmerade att köra mer energiffektivt än bränsledriva fordon. I bränsledriva fordon har den mänskliga föraren tendensen att använda gasen-och-bromsen oftare än nödvändigt, vilket i sin tur förbrukar mer bränsle än det behövs. Däremot är autonoma fordon programmerade för att uppnå maximal effektivitet hela tiden. Autonoma bilar har dessutom en stor potential att minska miljöutsläppen upp till 90% [32]. Sverige har som mål att år 2050 inte ha några nettoutsläpp av växthusgaser. Detta innebär att utsläpp från olika svenska källor, inklusive transportsektorn måste bli noll. [33] Vilket gör det elektriska autonoma fordonet till ett miljövänligare val och ett nödvändigt hållbar lösning på sikt.

Det finns dock en risk att autonoma fordon kommer att öka energiförbrukningen upp till 200%. Den största nackdelen är att man förmodligen kommer köra flera mil med bilen då bilresan kommer att bli bekvämare och dessutom mycket enklare då det ej kommer behövas en aktiv förare. Vi kommer kanske att ske en ändring i körmönster och beteenden. Bilägarna kommer att göra resor som de för närvarande inte skulle göra för att bilkörningen kommer att bli lättare. Även om pendlare gynnas av att de kan utföra andra uppgifter under den tid de normalt skulle köra, kan detta påverka miljön negativt.

Tekniken i autonoma bilar skulle dock kunna hjälpa till att planera och förbättra trafikflödet, och inte längre låta det att ske slumpvis. Detta skulle minska köbildningar och tomgångskörning. Den autonoma bilen som kan välja de mest energieffektiva vägarna och reser snabbare har störst chans att påverka miljön positivt. [34]

Referenser

[1]. NHTSA's National Center for Statistics and Analysis. "Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey." A brief Statistical Summary. Hämtades 2019-05-25, https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115

[2]. Union of Concerned Scientists, Science for a healthy planet and Safer World. "Self-Driving Cars Explained.How do self-driving cars work-and what do they mean for the future?" Hämtades 2019-05-25. https://www.ucsusa.org/clean-vehicles/how-self-driving-cars-work

[3]. Automated Guided Vehicles,Niklas Carlsson Filip Norberg Examensarbete i logistik, En förstudie på Rusta AB:s centrallager,Termin: VT18. Hämtades 2019-05-25 http://www.diva-portal.org/smash/get/diva2:1214700/FULLTEXT01.pdf

[4]. Figur 1.1, Illustration av olika möjliga uppkopplingar av autonoma fordon, Center for Advanced Automotive Technology. Hämtades 2019-05-25 http://autocaat.org/Technologies/Automated_and_Connected_Vehicles

[5]. Center for Advanced Automotive Technology, Hämtades 2019-05-25 http://autocaat.org/Technologies/Automated_and_Connected_Vehicles/

[6]. Figur 1.2, National Highway Traffic Safety Administration, Hämtades 2019-05-25 https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#issue-road-self-driving

[7]. What is a Computer Network? - Definition from Techopedia. Hämtades 2019-05-25, https://www.techopedia.com/definition/25597/computer-network

[8]. Figur 2.1: Illustration av ett datornätverk http://47.96.156.55/Public/Uploads/library/530e1cfd09b1e.pdf

[9]. TCP/IP Illustrated. Hämtades 2019-05-25 http://47.96.156.55/Public/Uploads/library/530e1cfd09b1e.pdf

[10]. Figur 2.2: Illustration av internetprotokollet http://47.96.156.55/Public/Uploads/library/530e1cfd09b1e.pdf

[11].TCP/IP Illustrated, Volume 1, The Protocols, W.Richard Stevens. Hämtades 2019-06-01

http://justpain.com/eBooks/Networking/TCP-IP/TCP-IP%20Illustrated%20-%20Vol%201.pdf

[12]. DNS, TCP/IP Illustrated. Hämtades 2019-05-25
http://47.96.156.55/Public/Uploads/library/530e1cfd09b1e.pdf

[13].What Is a Socket? (The Java™ Tutorials > Custom Networking > All
Hämtades 2019-05-25
https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[14]. Introduktion av Raspberry Pi, Hämtades 2019-05-25
https://projects.raspberrypi.org/en/projects/raspberry-pi-getting-started

[15]. About Python™ | Python.org. Hämtades 2019-05-25,
https://www.python.org/about/

[16]. ZeroMQ Feature List - zeromq. Hämtades 2019-05-25
http://zeromq.org/docs:features

[17]. PUB/SUB - Learning ØMQ with pyzmq - Read the Docs. Hämtades 2019-05-25
https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/pubsub.html

[18]. Figur 3.1: Illustration av PUB/SUB. Hämtades 2019-06-21
https://realtimeapi.io/author/justinucd/page/3/

[19]. Figur 3.2: Illustration av PUB/SUB. Hämtades 2019-06-21
https://www.codeproject.com/Articles/488207/ZeroMQ-via-Csharp-Introduction

[20]. Client / Server - Learning ØMQ with pyzmq - Read the Docs. Hämtades 2019-05-25
https://learning-0mq-with-pyzmq.readthedocs.io/en/latest/pyzmq/patterns/client_server.html

[21]. REQ/REP Pattern figur. Hämtades 2019-06-21
https://www.codeproject.com/Articles/488207/ZeroMQ-via-Csharp-Introduction

[22]. Protocol Buffers - Google Developers. Hämtades 2019-05-25
https://developers.google.com/protocol-buffers/docs/overview

[23]. CAN protocol. Understanding the controller network protocol. Hämtades 2019-06-24
https://www.engineersgarage.com/article/what-is-controller-area-network

[24]. Figur 3.4. CAN-kommunikation, Hämtades 2019-05-25
https://www.engineersgarage.com/article/what-is-controller-area-network

[25] ADAS. Hämtades 2019-05-25
https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/ersosynthesis2016-adas15_en.pdf

[26] Download NOOBS for Raspberry Pi. Hämtades 2019-06-17
https://www.raspberrypi.org/downloads/noobs/

[27].Raspberry Pi 2 & 3 Pin Mappings, Hämtades 2019-05-28
https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/pinmappings/pinmappingsrpi

[28]. How do RBG Leds work. Hämtades 2019-05-25
https://randomnerdtutorials.com/electronics-basics-how-do-rgb-leds-work/'

[29]. Figur 6.2, RGB_LED diod, Hämtades 2019-05-25
http://el.st/?rgb_ljusstake

[30]. Figur 6.3: Kretskopplingen för en RGB-LED diod på en kopplingsplatta.
Hämtades 2019-05-25 http://el.st/?rgb_ljusstake

[31]. An Intro to Threading in Python – Real Python. Hämtades 2019-05-25
https://realpython.com/intro-to-python-threading/

[32]. How self-driving cars could impact the environment – Blue & Green Tomorrow.
Hämtades 2019-05-25
https://blueandgreentomorrow.com/environment/self-driving-cars-could-impact-environment/

[33]. Innebörden av nettoutsläpp 2050, Sveriges Riksdag, Miljöminister Lena Ek.
Hämtades 2019-05-25
https://www.riksdagen.se/sv/dokument-lagar/dokument/svar-pa-skriftlig-fraga/inneborden-av-nettoutslapp-2050_GZ12294

[34]. Self-driving cars and their environmental impact, Payton Chang.
Hämtades 2019-05-25
http://large.stanford.edu/courses/2017/ph240/chang-p2/

Appendix 1

GPIO Pin Overview

The following GPIO pins are accessible through APIs:

GPIO#	Power-on Pull	Alternate Functions	Header Pin
2	PullUp	I2C1 SDA	3
3	PullUp	I2C1 SCL	5
4	PullUp		7
5	PullUp		29
6	PullUp		31
7	PullUp	SPI0 CS1	26
8	PullUp	SPI0 CS0	24
9	PullDown	SPI0 MISO	21
10	PullDown	SPI0 MOSI	19
11	PullDown	SPI0 SCLK	23
12	PullDown		32
13	PullDown		33
16	PullDown	SPI1 CS0	36
17	PullDown		11
18	PullDown		12
19	PullDown	SPI1 MISO	35
20	PullDown	SPI1 MOSI	38
21	PullDown	SPI1 SCLK	40
22	PullDown		15
23	PullDown		16
24	PullDown		18
25	PullDown		22
26	PullDown		37
27	PullDown		13
35*	PullUp		Red Power LED
47*	PullUp		Green Activity LED

* = Raspberry Pi 2 ONLY. GPIO 35 & 47 are not available on Raspberry Pi 3.