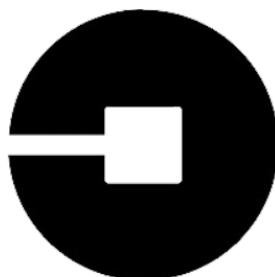


Kandidatuppsats i elektronik
Institutionen för systemteknik, Linköpings universitet, 2021

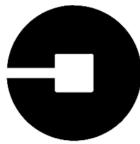
Konstruktion av en autonom taxibil

Design of an Autonomous Taxicab

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl



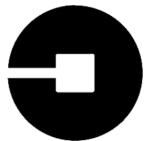
UNTER

**UNTER****Projektidentitet**Grupp E-post: **jonca673@student.liu.se**Hemsida: **<http://wwwisy.liu.se/edu/kurs/TSEA56/>**Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: **anders.p.nilsson@liu.se**Handledare: Olov Andersson
Tfn: +46 13282658
E-post: **olov.andersson@liu.se**Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: **mattias.krysander@liu.se****Projektdeltagare**

Namn	Ansvar	E-post
Jonathan Carlin	Projektleddare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Datorseendeansvarig (DAT)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se

**UNTER****INNEHÅLL**

1	Inledning	1
2	Problemformulering	1
2.1	Datorseende	2
2.2	Reglering	2
2.3	Hinder	2
2.4	Integration av moduler	2
3	Kunskapsbas	3
4	Genomförande	3
4.1	Före projektstart	4
4.2	Under projektet	4
4.3	Efter projektet	5
5	Teknisk beskrivning	6
5.1	Kommunikationsmodul	7
5.2	Styrmodul	7
5.3	Sensormodul	8
5.4	Tekniska utmaningar	9
6	Resultat	9
6.1	Bilen	9
6.2	Bilens prestanda	10
6.3	Testning av bilen	10
7	Slutsatser	11
7.1	Reflektion över projektarbetet	11
7.2	Vidareutveckling	12
7.3	Utveckling av uppgiften	12
A	Kravspecifikation	14
B	Banspecifikation och tävlingsregler	38
C	Systemskiss	47
D	Projektplan	61
E	Designspecifikation	82
F	Förstudie Kommunikation	112
G	Förstudie Styrning	128
H	Förstudie Sensor	149
I	Teknisk Dokumentation	167
J	Användarmanual	226
K	Efterstudie	237



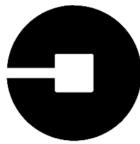
Autonom taxibil

2021-07-09

UNTER

DOKUMENTHISTORIK

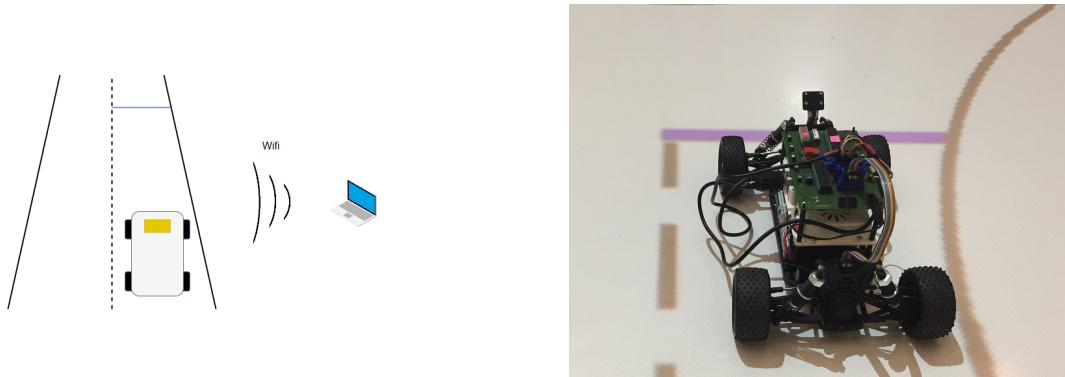
Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2019-05-21	Version 1.0	Grupp 9	N.S.



UNTER

1 INLEDNING

Uppdraget var att konstruera en autonom taxibil som i ett känt vägnät kunde utföra köruppdrag. Givet en karta av ett känt vägnät, en startposition samt ett antal destinationer var köruppdraget att besöka alla de givna destinationerna genom att autonomt navigera i vägnätet. Bilen navigerade med hjälp av datorseende. Vägnätet bestod av vägar med två filer, rondeller, stopplinjer samt hinder enligt Appendix B.



Figur 1: Översikt av bilen.

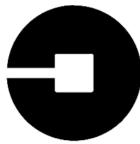
För att navigera i vägnätet var bilen utrustad med en kamera samt ett antal sensorer som utnyttjades av processorerna på bilen. Kamerans uppgift var att överföra bilder till en Raspberry Pi 3 Model B (RPI) som detekterade väglinjer med datorseendealgoritmer. De andra sensorerna, två halleffektswitchar samt en ultraljudsensor, användes för beräkning av körvstånd respektive detektion av hinder.

Under den första fasen hade projektgruppen många framgångar. RPI:n kunde snabbt kopplas upp mot gruppens persondator så trådlös kommunikation mellan en persondator och bilen kunde upprättas tidigt. Mycket av de enklare momenten, som manuell styrning, blev avklarade under projektets andra vecka. Gruppens största utmaning under projektet var att få datorseendet att fungera i salen Visionen där demonstrationen hölls. Datorseendet hade kalibrerats och testats i labbmiljö innan tester i Visionen började och gruppen blev då tvungna att tänka om mycket kring datorseendet då den nya miljön presenterade andra problem för datorseendet än de som återfanns i labbmiljön. Vid projektets avslutande veckor hade gruppen lyckats skapa ett datorseende för Visionen som var tillräckligt robust.

I detta dokument presenteras projektets problemformulering, genomförande och resultat samt de slutsatser som kunde dras. Alla dokument som författats under projektets utförande finns i Appendix.

2 PROBLEMFORMULERING

För att rikta utvecklingen av bilen så att den skulle kunna utföra köruppdrag ställdes en uppsättning krav upp. Bilen var tänkt att autonomt navigera i ett projicerat vägnät i salen Visionen. Vid navigation i vägnät var det krav på att bilen inte körde utanför vägen, att bilen stannade för stopplinjer samt att bilen ej slingrade sig fram vid körning. Bilen behövde således vara kapabel att detektera sin position relativt vägen och kunna detektera stopplinjer. För att förhindra att bilen slingrade sig fram vid körning ställdes krav på en robust regleringsalgoritm. För att minimera körsträckan



UNTER

för bilen under köruppdrag ställdes krav på att bilen utifrån en karta av vägnätet löste ett kortaste väg-problem för att beräkna körsträckan för köruppdraget. För en fullständig lista av alla bilens krav se Appendix A.

I nedanstående avsnitt diskuteras de krav projektgruppen ansåg vara mest utmanande och intressanta.

2.1 Datorseende

För att färdas i vägnätet utan att köra utanför vägen krävdes en robust linjedetektion för att detektera vägens väglinjer. Väglinjerna var nödvändiga för att beräkna bilens position relativt vägen. Bilen var ett realtidssystem vilket medförde att bildbehandlingen för detektion av väglinjer behövde utföras på kortast möjliga tid då bilen var i ständig rörelse. Om tiden mellan två linjedetektioner var lång försämrades uppskattningen för bilens position relativt vägen. Optimeringen av datorseendealgoritmernas exekveringstid var ett intressant problem då det var nödvändigt att göra en avvägning av insamlad information gentemot exekveringstid.

2.2 Reglering

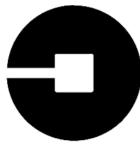
En regleringsalgoritm skapades för att bilen skulle kunna följa vägen utan risk för avåkning. Algoritmen var även tvungen att kunna hantera rondeller. Regleringsalgoritmen var mer utmanande än förväntat då regleringen lämpligen skedde utifrån de detekterade väglinjerna som inte var helt exakta och i vissa situationer missades av linjedetektionen. Den första versionen av regleringsalgoritmen krävde att båda väglinjer detekterades samtidigt. Tidiga tester visade dock att regleringsalgoritmen var bristfällig i situationer där datorseendet endast kunde detektera en enskilda linje, vilket ofta kunde ske i kurvor. Bristen åtgärdades genom att regleringsalgoritmen skrevs om så att den kunde reglera på endast en väglinje i taget. I första hand reglerade bilen efter höger väglinje, eftersom datorseendet såg den solida linjen mer pålitligt, och i andra hand vänster väglinje. Denna reglering var endast bristfällig i situationer där datorseendet inte kunde detektera någon väglinje, vilket var en mycket mer sällsynt situation än situationerna där endast en enskilda väglinje kunde detekteras. I ett tidigt skede identifierades att separata regleringsalgoritmer skulle krävas för att reglera på vanlig väg och i rondeller.

2.3 Hinder

För att detektera och stanna för hinder utnyttjades en avståndsmätare för att avgöra om det detekterade hindret var tillräckligt nära bilen. Om så var fallet skulle ett stopp utföras så att bilen stannade tills hindret var borta. En ultraljudssensor valdes till detta efter överläggning med handledare. Fördelen med att använda en ultraljudssensor istället för en lasersensor var att den skickade ut ljud i form av en kon vilket gjorde att den med god marginal kunde detektera hinder även vid oscillerande beteende från bilen samt i kurvor där bilens främre del inte var precis riktad mot hindret.

2.4 Integration av moduler

De olika modulerna kommunicerade med varandra genom UART (Universal Asynchronous Receiver/Transmitter) bussar, som är en typ av asynkron kommunikation. Data som t.ex. styrutslag, gaspådrag och sensordata från och till kommunikationsmodulen och AVR-processorerna förmedlades med busskommunikation. Den stora utmaningen med busskommunikationen var att använda den både till att skicka koder som tolkades olika av olika AVR-processorer och mädata som erhölls från båda mikroprocessorerna. Att få till ett samspel av väntan och svar i kommunikationen mellan moduler var krävande men givande arbete då denna kommunikation spelade en central roll för hela systemet.



UNTER

3 KUNSKAPSBAS

Utöver den redan nämnda RPI:n användes två mikroprocessorer av typen ATmega1284P i projektet. Den ena användes som grunden för styrmodulen som ansvarade för styrning av motor och styrservo. Den andra användes som grunden för sensormodulen som ansvarade för insamling av data från avståndssensor och halleffektswitchar. Databladet för mikroprocessorn användes flitigt under projektets gång för att korrekt programmera önskad funktionalitet. [1]

Ultraljudssensorn SRF04 användes i projektet för detektion av hinder och det datablad på ISY:s databladsserver Vanheden användes för att kontrollera hur sensorn fungerade och användes. [2]

Halleffektswitchen A1120 användes i projektet för beräkning av färdat avstånd. Databladet som användes för sensorn finns på ISY:s databladsserver Vanheden. [3]

För kommunikation mellan bilens olika moduler använde sig projektgruppen av UART kablar för busskommunikation. Projektgruppen använde sig av databladet för ATmega1284P vid programmering av UART kablarna då det innehöll mycket bra information och var anpassad efter den AVR som gruppen använde. [1]

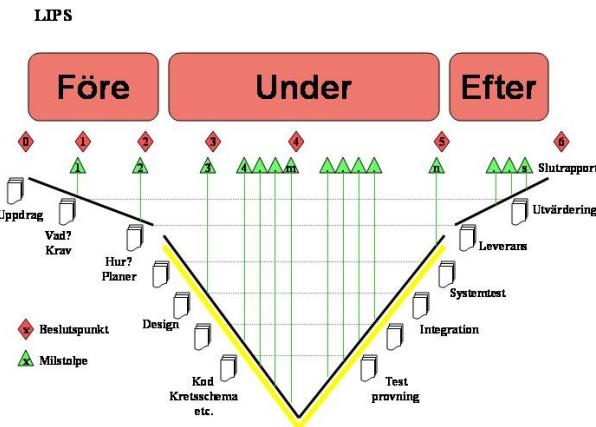
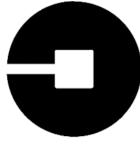
Det datorseende som implementerades i projektet använde sig av det öppna källkodsbiblioteket OpenCV. Då ingen i projektgruppen tidigare arbetat med datorseende eller OpenCV användes dokumentationen flitigt under projektets gång. [4]

Projektmedlemmarna hade tidigare läst kurser i tekniska ämnen som reglertechnik, programmering (i programmerings-språket C++) och datorteknik som bildat en kunskapsbas inom gruppen att tillgå. I de områden där projektgruppen saknat kunskap har de själva utbildat sig genom t.ex. dokumentation för programmet samt officiella datablad för processorerna.

Utöver ovanstående datablad och dokumentation blev projektgruppen tilldelad en handledare från ISY som bidrog med experthjälp i projektet. Handledaren användes flitigt under projektets gång för att rådfråga om implementations-idéer samt hjälpt till de gånger projektgruppen stött på problem som i rimlig tid inte kunde lösas av gruppen själv.

4 GENOMFÖRANDE

Projektet har bedrivits enligt LIPS-mallen och de moment den medför. I följande rubriker presenteras hur projektets olika faser uppställda i projektplanen, se Appendix D, har genomförts. Figur 2 är en beskrivande bild över hur ett projekt genomförs med hjälp av LIPS-modellen.



Figur 2: Översikt av Lips-modellen

4.1 Före projektstart

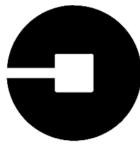
Projektet påbörjades med att beställaren tillhandahöll ett projektdirektiv med riktlinjer för projektet. Utifrån det givna projektdirektivet producerades en kravspecifikation med uppställda krav för bilens funktionalitet. Kraven tilldelades olika prioritetsnivåer på en skala från 1 till 3. Krav med prioritet 1 var baskrav som behövde uppfyllas, prioritet 2 var börkrav som uppfylldes i mån av tid och prioritet 3 var extrakrav som kunde ses som utvecklingsmöjligheter för bilen.

Efter att beställaren godkänt kravspecifikationen påbörjades en projektplan, en systemskiss och en tidplan. Projektplanen detaljerade hur projektgruppen planerade att få projektet genomfört med goda resultat. Aktiviteter och milstolpar togs fram för att strukturera upp projektet och få en god översikt av projektets olika delmoment. Systemskissen beskrev hur projektgruppen planerade att konstruera bilen och bilens olika moduler. Placering av bilens olika moduler bestämdes och vad dessa moduler behövde för att uppfylla de krav som ställts upp i kravspecifikationen. Tidplanen beskrev hur projektmedlemmarnas tid planerades att distribueras samt i vilken ordning och den uppskattade tiden för projektets aktiviteter.

I samband med att projektplan, systemskiss och tidsplan skrevs påbörjades ett antal förstudier för att undersöka vilka möjligheter som existerade för att genomföra projektets delar. Total skrevs tre förstudier. En för trådlös- och busskommunikation, en för reglering av framhjulstylda fordon samt en för datorseendealgotimer för att detektera väglinjer. Se Appendix F, G respektive H.

4.2 Under projektet

I samband med att beställaren godkände den projektplan, systemskiss och tidplan projektgruppen tagit fram påbörjades projektet. Innan projektgruppen påbörjade det praktiska arbetet skapades en designspecifikation. Designspecifikationen gav en ingående beskrivning av bilen och baserades på de tre förstudier som utförts under projektets före fas. Designspecifikationen blev underlag för det praktiska utförandet av projektet. Att detaljerat beskriva den planerade designen av bilen i förväg underlättade genomförandet då projektgruppen fick bra översikt av hur bilen och bilens



UNTER

moduler planerades att konstrueras. Designspecifikationen beskrev den planerade hårdvarumässiga och mjukvarumässiga konstruktionen av bilen och bilens moduler. Beskrivningar för de olika modulerna skrevs på komponentnivå och beräkningar utfördes för att beräkna toleransnivåer för bilens olika delsystem.

Efter att designspecifikationen blivit godkänd påbörjades det praktiska arbetet inom projektet. Bilen var modulärt uppbyggd med tre olika moduler som specificerats av beställaren i projektdirektivet. Bilen hade därmed tre distinkta delsystem som kunde utvecklas separat. För att parallelisera arbetet i så stor mån som möjligt kvitterade projektgruppen ut mer hårdvara än vad bilen behövde. Detta gjorde det möjligt för projektgruppen att utveckla bilens tre delsystem samtidigt på separat hårdvara. Efter att bilens olika delsystem var klara kopplades delsystem ihop till ett enda modulärt uppbyggt system med busskomunikation. När bilens olika delsystem kopplats ihop till ett enda system påbörjades utvecklingen för att navigera i ett vägnät. Det var nödvändigt att göra utförliga tester för bilens datorseende och reglering då miljön i salen Visionen var mer annorlunda än projektgruppen förväntat.

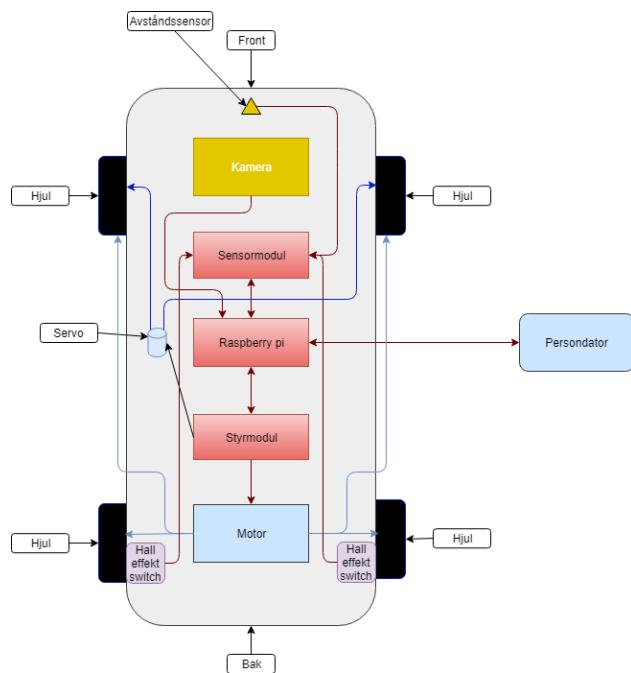
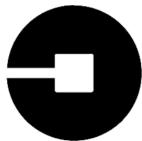
Vid det praktiska utförandet av projektet låg huvudansvaret för de olika modulerna på de personer som skrivit förstudie för respektive modul. Projektgruppen ansåg att alla i gruppen borde ha viss kunskap om hur de olika modulerna fungerade och såg därmed till att projektmedlemmarna fick arbeta med alla av bilens olika delsystem. Till största del följde arbetet den tidplan som tagits fram under projektets före fas men då projektgruppen inte hade mycket erfarenhet av projekt av denna storlek tidigare underestimerades tiden för en del av projektets aktiviteter. I de fallen uppdaterades tidplanen för att ta hänsyn till förändringen av aktiviteters allokerade tid.

4.3 Efter projektet

Efter det att bilen var färdigkonstruerad och projektets praktiska arbete avslutats skrevs en teknisk dokumentation för bilen. Den tekniska dokumentationen redogjorde för hur bilen var konstruerad samt vilka designbeslut projektgruppen tagit vid konstruktion av bilen.

En användarmanual för bilen skrevs för de som eventuellt skulle använda bilen efter att projektet avslutats. Användarmanualen beskrev hur bilen används på ett sådant sätt att bilens fulla funktionalitet utnyttjas.

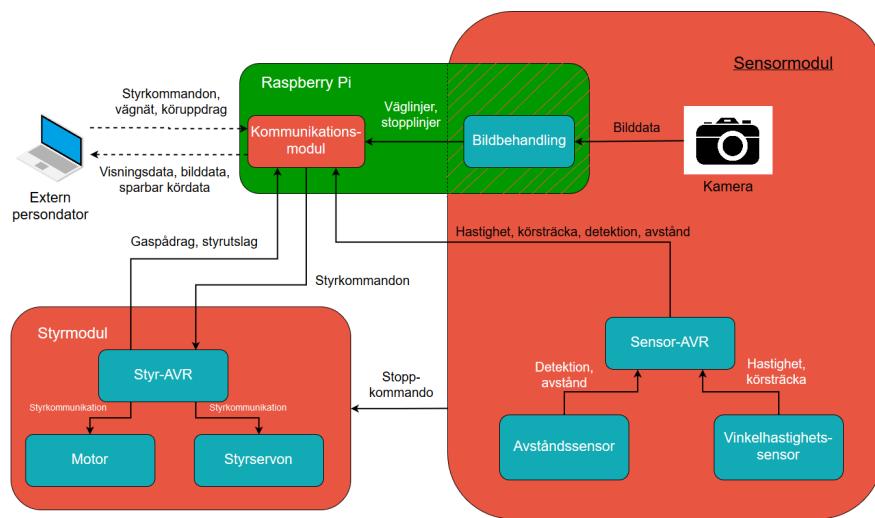
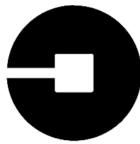
Vid projektets avslut verifierades att alla de uppställda kraven i kravspecifikationen, se Appendix A, uppfylldes av den konstruerade bilen. En tävling mot liknande bilar konstruerad utifrån samma projektdirektiv anordnades av beställaren för att undersöka hur bilarna presterade gentemot varandra. Till sist skrev projektgruppen en efterstudie för att utvärdera det utförda arbetet. I efterstudien reflekterade projektgruppen på vad som gått bra i projektet och vad som kunde ha gått bättre samt hur den konstruerade bilens förbättringsmöjligheter såg ut för vidareutveckling av bilen.



Figur 3: Översikt av bilen med moduler.

5 TEKNISK BESKRIVNING

Bilen består av ett chassi på vilket bilens delsystem är monterade enligt figur 3. Delsystemen består av tre moduler, en kommunikationsmodul, en styrmodul och en sensormodul som är sammankopplade enligt figur 4. I följande rubriker ges en beskrivning av bilens moduler och de tekniska utmaningar projektgruppen har överkommit under projektets utförande.



Figur 4: Illustration av hur bilens moduler är sammankopplade. OBS regleralgoritmen från styrmodul ligger på Pi.

5.1 Kommunikationsmodul

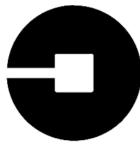
Kommunikationsmodulen ansvarade för all kommunikation med externa system och den interna kommunikationen mellan bilens delsystem. Kommunikation med utomstående system skedde trådlöst via IEEE 802.11-protokollet (mer känt under marknadsföringsnamnet WiFi) medan den interna kommunikationen skedde med databussar av typen USART (Universal synchronous and asynchronous receiver-transmitter), där den synkrona möjligheten inte användes. Valet om WiFi samt USART för kommunikation baserade sig på den förstudie som gjordes i projektets förefas, se Appendix F, där de olika alternativen för kommunikation undersöktes.

Kommunikationsmodulen tog emot data gällande dåvarande sensorvärdet och detekterade väglinjer från sensormodulen samt dåvarande gaspådrag och styrutslag från styrmodulen. Data från sensormodulen användes för regleringen av bilen och data från styrmodulen lagrades för att efter test undersöka förändringar över tid. Kommunikationsmodulen skickade data för nytt styrutslag uträknat av regleralgoritmen till styrmodulen för att uppdatera styrervots styrutslag. Ingen kommunikation skedde från kommunikationsmodulen till sensormodulen.

Då all kommunikation hanterades av kommunikationsmodulen fick modulen en central roll och kom att agera som bilens beslutsfattande huvudmodul. Utöver kommunikation skötte modulen beräkningen av kortaste vägen för köruppdrag och agerade som beslutscentrum i de fall då bilen behövde ta ett beslut som exempelvis vilken utfart som skulle tas ur en rondell.

5.2 Styrmodul

Styrmodulen ansvarade för kontroll av den motor och det styrervo som var monterat på bilens chassi. För att kontrollera den spänning motorn och styrervo erhöll utnyttjades den kapabilitet för pulsbreddsmodulering som fanns på mikroprocessorn Atmega1284P. Genom att variera hur lång tid signalen var hög var 20:e millisekund kunde den spänning som skickades ut varieras. För en mer detaljerad beskrivning se Appendix I.



UNTER

Det vägnät som skulle navigeras innehöll svängar och rondeller som bilen behövde klara av utan att köra utanför vägen. För att lösa problemet behövde bilen en reglering som klarade av dessa situationer. Då styrmodulen ansvarade för all bilens rörlighet var det naturligt att ge styrmodulen ansvaret för regleringen av bilen. Regleringen använde sig av en PD-regulator då förstudien för styrmodulen, se Appendix G, kommit fram till resultatet att en PD-regulator var den mest effektiva lösningen på problemet.

Styrmodulens reglering använde sig av bilens position relativt vägens väglinjer för att beräkna reglerfelet. Genom att få bilens position relativt vägen att gå mot ett satt referensvärde klarade bilen av att åka i vägnätet utan att slingra sig fram.

Styrmodulen kunde få en stoppsignal från sensormodulen i de fall sensormodulen detekterat ett hinder för nära bilen. I de fall en stoppsignal erhölls stoppades bilens gaspådrag och styrutslag genom att ställa motorns gaspådrag och styrervots styrutslag i neutralt läge.

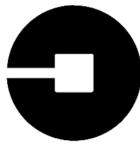
5.3 Sensormodul

Sensormodulen ansvarade för de sensorer som monterades på bilen. Sensorerna var en ultraljudssensor för hinderdetektion och två halleffektsswitchar för mätning av körsträcka. Även den kamera och det datorseende kameran använde inkluderades i sensormodulen. Kameran kom monterad på det chassi som tillhandahölls och gruppen valde att inte göra några ändringar kamerans placering. Valet av sensorerna saknade underliggande beslutsunderlag från förstudien som fokuserade på datorseende utan baserade sig istället på en rekommendation från den handledare projektgruppen blivit tilldelad.

Huvuduppgiften för sensormodulen var att detektera väglinjer och stopplinjer då bilen utförde köruppdrag. Detektionen av väglinjer använde sig av en Raspberry Pi kameramodul version 2 och det datorseende som implementerades använde sig av biblioteket OpenCV. Genom att utföra ett antal bildbehandlingsoperationer på bilder tagna av kameran kunde väglinjer detekteras och isoleras från resten av bilden. Utifrån de detekterade väglinjerna beräknades ett antal parametrar för bilens reglering som skickades vidare till kommunikationsmodulen. Detektion av stopplinjer utfördes samtidigt som detektion av väglinjer. Då stopplinjerna var i huvudsak horisontella på de bilder kameran tog av vägen isolerades detekterade horisontella linjer. Om tillräckligt många horisontella linjer detekterades klassificerades det som en stoppline och informationen att en stoppline detekterats skickades till kommunikationsmodulen.

Sensormodulens andra uppgift var att samla in data från de monterade sensorerna och översätta insamlad data. Den insamlade datan representerade inte direkt en fysikalisk storhet utan behövde processeras för att dra slutsatser om vad datan betydde. För en mer detaljerad beskrivning av hur insamlad sensordata översattes till termer av avstånd och färdsträcka se Appendix I.

Sensormodulen tar till största del inga beslut själv utan skickade endast den tolkade sensordatan och resultatet från datorseendet till bilens kommunikationsmodul som ansvarade för beslutstagande. Det enda beslut sensormodulen själv tog var då bilen skulle stanna för detekterat hinder. När ett hinder detekterades tillräckligt nära bilen skickade sensormodulen ut en stoppsignal till bilens styrmodul.



UNTER

5.4 Tekniska utmaningar

Vägnätet som projicerades i salen Visionen var en större utmaning för datorseendet än projektgruppen förväntat sig. I Visionen projicerades vägnät på golvet som var en blank vit yta. Den blanka ytan resulterade i att ljus från lampor i omkringliggande rum reflekterades på ytan in i kameralinsen. Då vägnätet projicerades på en vit yta var det omöjligt att få en klart svart färg utan den svarta färgen var en mer gråliknande blandning av rött, grönt och blått. Kombinationen av blänk och dålig svart färg introducerade svårigheter att skilja vägens linjer från störningarna av blänk. För att lösa problemet designades datorseendet sådant att en stor del av bilderna som innehåll störningar men inte väglinjer filtrerades bort. Detektionen av väglinjer använde från början en kantdetektionsmetod som använde gradienten för att detektera kanter. Metoden identifierade dock många störningar som kanter och ersattes av en metod som filtrerade bort alla pixlar ljusare än en viss satt gräns. Detta medförde att vägens linjer detekterades medan mängden störningar minskade.

Exekveringstiden av datorseendealgoritmen var en annan teknisk utmaning i projektet. Om exekveringstiden varit för lång hade regleralgoritmen erhållit uppdaterade parametervärden för bilens position relativt vägen vilket skulle ha resulterat i en försämrad reglering. För att optimera datorseendealgoritmen minimerades antalet operationer som bildbehandlingen utförde till endast de operationer som var nödvändiga för att detektera linjer i salen Visionen. Projektgruppen upptäckte tidigt att själv linjedetektionen som använde houghtransform tog majoriteten av tiden. Linjedetektionen var ett nödvändigt steg som inte kunde uteslutas. Istället fick projektgruppen optimera transformen själva genom att använda probabalistisk houghtransform och minska upplösningen av transformen. På så sätt sänktes exekveringstiden markant medan datorseendets funktionalitet bevarades.

Regleringsalgoritmen använde parametervärden uträknade av datorseendealgoritmen för att reglera bilens framfart på vägen. Då datorseendealgoritmens linjedetektion i alla fall inte kunde garantera att en linje detekterades uppstod det specialfall då endast en av linjerna eller fall då ingen av väglinjerna detekterades. För att överkomma problemet designade projektgruppen regleralgoritmen att reglera efter endast en linje, den detekterade vänstra eller den detekterade högra väglinjen. Väglinjerna tilldelades en prioritet beroende på dåvarande vägsegment. I vägnätet gällde högertrafik vilket innebar att i raksträckor och svängar var den heldragna sidolinjen, som var lättare att detektera, den högra väglinjen. I rondeller var både vänster och höger väglinje heldragna men den högra väglinjen hade avbrott vid rondellens utfarter. Utifrån ovanstående resonemang var det naturligt att prioritera reglerinn med högra väglinjen i raka och svängde vägsegment och att prioritera den vänstra väglinjer vid rondellkörning. I de fall bilen inte detekterade någon linje utfördes ingen reglering då ingen data fanns att tillgå för regleralgoritmen.

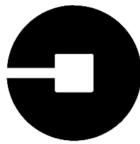
6 RESULTAT

I nedanstående avsnitt presenteras de resultat projektgruppen har uppnått under projektets genomförande.

6.1 Bilen

Bilen som konstruerats är en autonom taxibil som med datorseende kan navigera i ett vägnät för att utföra köruppdrag. Bilen har tre olika lägen för köring, ett manuellt, ett semi-autonoms och ett autonomt läge. Användaren väljer vilket läge bilen ska använda med terminalen då bilen är i *mode select* - fas.

Vid manuellt läge har användaren själv full kontroll över bilen och kontrollerar gaspådrag och styrutslag med tan-



UNTER

gentbordet.

I semi-autonomt läge kontrollerar bilen själv gaspådrag och styrutslag och användaren skickar istället ett eller flera styrkommandon till bilen. Efter att styrkommandon skickats till bilen utför bilden de givna kommandona och stannar sedan när alla kommandon har utförts.

I autonomt läge har användaren ingen kontroll över bilen utan kan endast skicka köruppdrag till bilen. När bilen har mottagit ett köruppdrag beräknar bilen den kortaste vägen för att utföra köruppdraget varav bilen sedan utför köruppdraget. När köruppdraget är klart stannar bilen och väntar på ett nytt köruppdrag från användaren. För en mer utförlig beskrivning av bilens olika körlägen se bilens användarmanual i Appendix J.

6.2 Bilens prestanda

Bilen har en reglering som klarar av att styra bilen i alla situationer som uppkommer i vägnätet utan att bilen slingrar sig fram. Regleralgoritmen kan även hantera de specialfall som uppstår då linjedektionen inte klarar av att detektera en väglinje med goda resultat.

Datorseendealgoritmen är robust nog att väglinjer detekteras i en stor majoritet av fallen även om bilderna tagna av vägen innehåller störningar från omkringliggande ljuskällor. Exekveringstiden för datorseendealgoritmen är tillräckligt kort för att regleralgoritmen alltid ska kunna reglera på relevanta parametrar för bilens position relativt de detekterade väglinjerna. Ultraljudssensorn för hinderdetektion klarar av att detektera hinder i alla de test projektgruppen genomfört och den skickade stoppsignalen fick bilen att stanna i alla testfall.

6.3 Testning av bilen

Testning av bilen och bilens delsystem har skett löpande under hela projektets genomförande. I projektets tidiga faser utfördes test av sensorvärdet samt gaspådrag och styrutslag vid projektgruppens station i salen Muxen 3. I projektets senare faser skedde alla test i salen Visionen eftersom det projicerade vägnätet ställde andra krav än testmiljön i muxen 3.

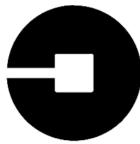
6.3.1 Ultraljudssensor

Test av sensormodulens ultraljudssensor skedde genom att mäta avståndet från sensorn till ett hinder och jämföra det uppmätta värdet mot det värde sensordatan representerade. Testerna utfördes i muxen och innan sensorn monterades på bilen och testades under körning användes en lysdiod för att kontrollera att sensorn fugerade som tänkt.

6.3.2 Styrservo och motor

Test av styrservot skedde genom att variera tillförd spänning till styrservot och kontrollera bilens styrutslag vid den givna spänningen. Tidigt i projektet upptäckte projektgruppen att styrservot monterat på bilens chassi inte stod riktat rakt då tillförd spänning var den rekommenderade standardinställningen för neutralt läge. Projektgruppen justerade tillförd spänning tills styrservot stod rakt och använde denna spänning som neutralt läge för styrservot.

Testning av den motor som var monterad på bilens chassi och tillhörande gaspådrag skedde på samma sätt som test



UNTER

för styrervot. Olika spänningar skickades till motorn och det resulterade gaspådraget undersöktes. Genom att variera spänningen undersöktes neutralt gaspådrag, hastigheter framåt samt bakåt.

6.3.3 Datorseende

Datorseendealgoritmens linjedektion testades genom att ta bilder av vägen och utföra linjedektion på bilderna. De detekterade väglinjerna ritades därefter ut på originalbilden och verifierades ha samma position och lutning som väglinjerna i originalbilden.

6.3.4 Reglering

Bilens reglering testades genom att placera bilen på olika positioner i vägnätet och tillåta bilen köra framåt. Parametrarna för regleralgoritmens primära och deriverande del ändrades tills dess att bilen kunde köra fram på alla delar av vägen utan att slingra sig fram eller köra utanför vägen.

6.3.5 Kortaste väg algoritm

Funktionaliteten hos bilens kortaste väg-algoritm testades genom att generera ett antal köruppdrag och skicka dem till kortaste väg-algoritmen. Resultatet kontrollerades genom att manuellt beräkna den kortaste vägen och verifiera att resultaten stämde med beräkningarna.

6.3.6 Test av hela systemet

När alla delsystem var färdigutvecklade och sammankopplade testades bilens funktionalitet i sin helhet genom att utföra köruppdrag i salen Visionen. Ett köruppdrag genererades och skickades till bilen varav bilen sedan utförde uppdraget. I de fall bilen inte klarade av att utföra köruppdraget undersöktes orsaken till felet och relevant kod eller hårdvara uppdaterades.

7 SLUTSATSER

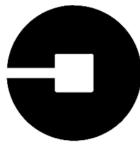
I följande avsnitt diskuteras gruppens slutsatser.

7.1 Reflektion över projektetarbetet

Projektet fortlöpte väldigt snabbt i början. Många lösningar som skapades under de första två veckorna blev väldigt robusta och lättanvända för de senare delarna. Gruppen lyckades även parallellisera arbetet på ett sätt som gjorde att alla medlemmar kunde arbeta med hög effektivitet. Implementationsmässigt så fungerar både datorseende och regleringen på ett robust och generellt sett vilket gör det enkelt att vidareutveckla projektet för t.ex. nya specifikationer på vägen.

Projektet gick över budget i form av tid, något som till stor del berodde på tillgängligheten till testsalen Visionen. Under de tidiga testerna närvarade också de flesta gruppmedlemmarna vilket i efterhand inte var den bästa idén då detta visade sig vara ineffektivt. Under senare tester begränsades närvaron vid tester till maximalt tre personer.

I projektets utförande hade projektgruppen problem med oidentifierat beteende från bilens mikrodatorer efter en avstängning som inte löstes förrän projektets senare stadier. Hade projektet gjorts igen hade problemet identifierats



UNTER

tidigare då det orsakade många irritationsmoment i projektet.

Hade projektet gjorts igen hade projektgruppen börjat testa bilens datorseende och reglering i salen Visionen mycket tidigare under projektets genomförande. I början testades bilen på vitt papper med svart tejp som linjer och när test av bilen senare skedde i Visionen var miljön så annorlunda att delar av datorseendet behövde skrivas om.

Bilen hade inget krav för möjlighet att ändra körhastighet vilket resulterade i ett hårdkodat gaspådrag. Detta hade biverkningen att bilens hastighet varierar något beroende på styrkan av det inkopplade batteriet. För att förhindra detta skulle projektgruppen ha implementerat reglering av bilens hastighet med hjälp av halleffektsswitcharna på hjulen om projektet genomfördes igen.

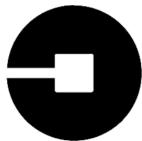
7.2 Vidareutveckling

Om projektgruppen hade haft mer tid till förfogande skulle gruppen velat skapa ett GUI för bilen för att visa relevant data från bilen under köruppdrag samt använda det till en smidigare inläsning av parametrarna till kortaste väg-algoritmen. En andra funktion projektgruppen hade velat implementera är att köra om hinder som detekteras på vägen istället för att stanna tills det att hindret lyftes bort.

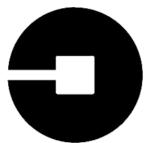
Till sist skulle projektgruppen velat få upp hastigheten på datorseendet och förbättrat reglering ännu mer så att bilen skulle kunnat köra i en högre hastighet i vägnätet.

7.3 Utveckling av uppgiften

En mer simpel utveckling av uppgiften som hade varit intressant att utforska är att inkludera saker som korsningar och "trafikljus" (kanske i form av färgskiftande stopplinjer) på vägnätet för att göra uppgiften så verklighetsliknande som möjligt. En mer tidskrävande och eventuellt mer riskfyllt utveckling hade varit att låta flera bilar utföra köruppdrag samtidigt på ett gemensamt vägnät. Risken att bilarna krockar med varandra är så klart överhängande vid denna implementation.

**UNTER****REFERENSER**

- [1] Atmel, “LiU, ISY, VanHeden,” <https://docs.isy.liu.se/pub/VanHeden/DataSheets/atmega1284p.pdf>, [Online; accessed may 15, 2019].
- [2] Unknown, “LiU, ISY, VanHeden,” <https://docs.isy.liu.se/pub/VanHeden/DataSheets/srf04.pdf>, [Online; accessed may 15, 2019].
- [3] ——, “LiU, ISY, VanHeden,” <https://docs.isy.liu.se/pub/VanHeden/DataSheets/DACar.pdf>, [Online; accessed may 15, 2019].
- [4] “OpenCV, development team,” <https://docs.opencv.org/3.0-beta/modules/refman.html>, [Online; accessed may 15, 2019].



UNTER

A KRAVSPECIFIKATION



UNTER

Autonom Taxibil

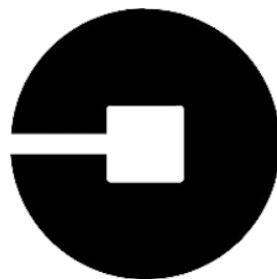
4 juni 2019

Kravspecifikation

Grupp 9

4 juni 2019

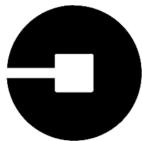
Version 1.1



UNTER

Status

Granskad	OM, JC, EM, DÖ, NS	2019-05-21
Godkänd		

**UNTER****Projektidentitet**Grupp E-post: jonca673@student.liu.se

Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

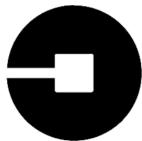
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Designansvarig (DES)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se



UNTER

INNEHÅLL

1	Inledning	1
1.1	Parter	1
1.2	Syfte och mål	1
1.3	Användning	2
1.4	Bakgrundsinformation	2
1.5	Definitioner	2
2	Översikt av systemet	2
2.1	Grov beskrivning av produkten	2
2.2	Produktkomponenter	3
2.3	Beroenden till andra system	3
2.4	Ingående delsystem	3
2.5	Avgränsningar	4
2.6	Designfilosofi	4
2.7	Generella krav	4
3	Kommunikationsmodul	5
3.1	Inledande beskrivning	5
3.2	Gränssnitt	5
3.3	Designkrav	6
3.4	Funktionella krav	6
4	Styrmodulen	6
4.1	Inledande beskrivning av styrmodulen	7
4.2	Externa gränssnitt	7
4.3	Designkrav	7
4.4	Funktionella krav	8
5	Sensormodul	8
5.1	Inledande beskrivning av sensormodulen	8
5.2	Externa gränssnitt	9
5.3	Designkrav	9
5.4	Funktionella krav	9
6	Persondator	10
6.1	Inledande beskrivning av persondatorn	10
6.2	Externa gränssnitt	10
6.3	Designkrav	10
6.4	Funktionella krav	11
7	Prestandakrav	12
8	Krav på vidareutveckling	13
9	Tillförlitlighet	13
10	Ekonomi	13
11	Krav på säkerhet	13
12	Leveranskrav och delleveranser	14
13	Dokumentation	15
14	Utbildning	16

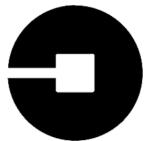


UNTER

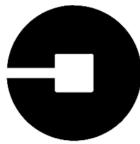
Autonom Taxibil

4 juni 2019

15 Kvalitetskrav	16
16 Underhållsbarhet	16
A Appendix	18

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda ändringar	Utförda av	Granskad
0.1	2019-01-30	Första utkast	Grupp 9	NS
1.0	2019-02-01	Första version.	Grupp 9	NS
1.1	2019-05-21	Första versionen uppdaterad	Grupp 9	JC



UNTER

1 INLEDNING

Detta dokument är kravspecifikationen för en autonom bil som ska levereras för projektkursen TSEA56. Dokumentet beskriver i detalj de krav systemet ska klara av vid leverans samt hur arbetet ska utföras. Den autonoma bilden ska klara av att köra runt i ett projicerat vägnät i lokalen Visionen. Bilen får inte kollidera med andra fordon eller hinder, bilen får heller inte bryta mot gällande trafikregler för högertrafik. Bilen ska autonomt ta sig till flera olika förbestämda destinationer på kortast möjliga väg. I slutet av kursen kommer en tävling hållas som utser den bästa bilen.

Systemet ska vara modulärt uppbyggd där man enkelt ska kunna byta ut en modul mot en annan utan att påverka funktionaliteten. Varje modul ska innehålla minst en processor.

Alla systemrelaterade krav beskrivs som tabeller enligt tabell 1. Den första kolumnen anger kravets nummer, den andra datumet då eventuella ändringar gjorts, den tredje vad kravet ska göra och den sista kolumnen, kolumn fyra, anger kravets prioritet. Kravnummer är löpande genom hela dokumentet och prioriteringen är numrerad från 1-3 där 1 är den högsta prioriteten och 3 den lägsta.

Tabell 1: Tabellutseende

Krav	Ändring	Beskrivning	Prioritet
1	Original	Alla dokument skall uppfylla instruktioner beskrivna i [1]	1

1.1 Parter

Denna kravspecifikation specificerar för beställaren de krav som ska uppnås av projektmedlemmarna vid slutet av projektet. Projektmedlemmarna kommer ta hjälp av en handledare vid viktiga designval i projektet. Projektets parter består av:

Beställare: Anders Nilsson (Institutionen för systemteknik (ISY)) vid Linköpings Universitet.

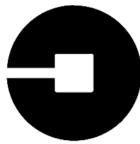
Handledare: Olov Andersson (Institutionen för systemteknik (ISY)) vid Linköpings Universitet.

Projektmedlemmar: Jonathan Carlin, Dennis Österdahl, Emil Mårtensson, Nicholas Sepp, Olof Mlakar.

1.2 Syfte och mål

Gruppens beställare vill undersöka möjligheten att konstruera en autonom bil och har därför anlitat tre olika grupper för att ta fram tre olika prototyper som ska användas som konstruktionsunderlag för den slutgiltiga produkten. De tre prototyperna ska demonstreras under en tävling så att beställaren kan utvärdera de olika konstruktionsalternativen. Gruppens syfte är att skapa en autonom bil som sedan ska tävla mot de andra gruppernas prototyp under gemensamt framtagna regler mellan grupperna och beställaren.

Gruppens mål är att ta fram en prototyp som är bättre än de konkurrerande gruppernas och sedan demonstrera detta under den överenskomna tävlingen.



UNTER

1.3 Användning

Gruppens prototyp ska, tillsammans med de andra gruppernas prototyper, användas som konstruktionsunderlag för att beställaren ska kunna skapa en autonom taxibil.

1.4 Bakgrundsinformation

Beställaren vill undersöka möjligheten att konstruera en autonom bil som kan utföra köruppdrag. Projektets uppdrag är att ta fram en prototyp för att undersöka hurvida detta är möjligt.

1.5 Definitioner

Nedan följer beskrivning av de begrepp och förkortningar som används i detta dokument.

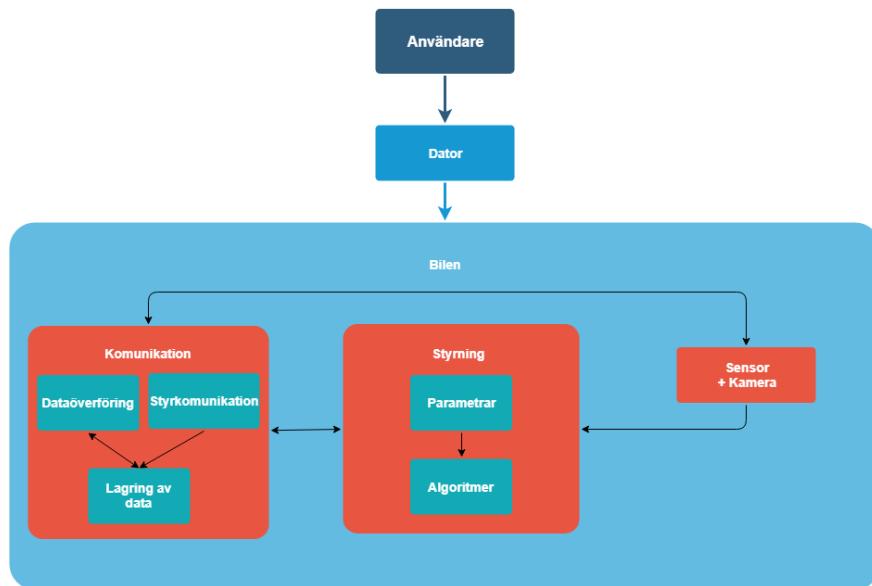
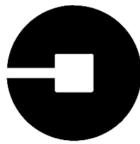
- BP2: Beslutspunkt 2, start av utförandefasen.
- BP5A: Beslutspunkt 5A, demonstration av baskrav
- BP5B: Beslutspunkt 5B, verifiering av slutkrav.
- Krav 1: Baskrav, de krav som har högst prioritet och ska vara uppnådd vid BP5A.
- Krav 2: Slutkrav, de krav som har näst högst prioritet och ska vara uppnådd vid BP5B.
- Krav 3: Extrakrav, de krav som har lägst prioritet och ska uppnås vid mån av tid.
- Demonstration: I slutet av projektet kommer en demonstration av produkten att hållas i form av en tävling mellan projektgrupperna.
- Chassi: Bilchassi som tillhandhålls av ISY. Chassit kommer förmonterat med hjul, hjulaxlar, varvräknare, elektriskmotor, m.m.
- Bilen: Systemprototypen - den autonoma bilen.

2 ÖVERSIKT AV SYSTEMET

Den autonoma bilen ska konstrueras med hjälp av tre separata moduler, kommunikations-, sensor- och styrmodul. Dessa moduler utgör systemet och figur 1 visar hur dessa samspelear.

2.1 Grov beskrivning av produkten

Produkten kommer att vara en autonom bil vars roll är att köra kortast möjliga väg till angivna destinationer i ett projicerat vägnät. Användaren ska kunna köra bilen autonomt, semi-autonomt eller fjärrstyrta.



Figur 1: En översikt av systemet

2.2 Produktkomponenter

De produktkomponenter som ingår i leveransen är en autonom bil, en demonstration, en teknisk dokumentation, en framläggning samt opposition, en efterstudie och en sammanställande rapport som dokumenterar projektet i sin helhet.

De komponenter som gruppen planerar att använda för bilen är följande:

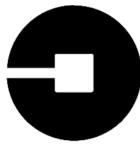
- Chassi
- Raspberry pi
- Persondator
- Vidvinkelkamera
- Avståndssensor

2.3 Beroenden till andra system

Bilen är beroende av en separat dator som kan kommunicera med bilen för att ge kommandon. Kommunikationen mellan datorn och bilden ska ske trådlöst.

2.4 Ingående delsystem

Bilen består av tre delsystem: en kommunikationsmodul, en styrmodul och en sensormodul (+kamera). Figur 1 visar dessa delsystem. De olika modulerna beskrivs mer omfattande i avsnitt 3-5.



UNTER

2.5 Avgränsningar

Bilen är inte designad för att kunna köra på ojämna underlag som förväntas i en verlig miljö. Bilen klarar inte av att identifiera hinder som är lägre än den minsta höjd specificerad i appendix A.

2.6 Designfilosofi

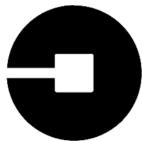
Bilen konstrueras med fokus på robusthet och pålitlighet. Bilen ska vara tillförlitlig och säkert kunna färdas till angivna destinationer. Bilen är modulärt uppbyggd så delsystem lätt kan bytas ut för uppdatering utan att påverka hela systemet.

2.7 Generella krav

Tabell 2 visar de generella kraven som ställs på systemet.

Tabell 2: Generell kravlista

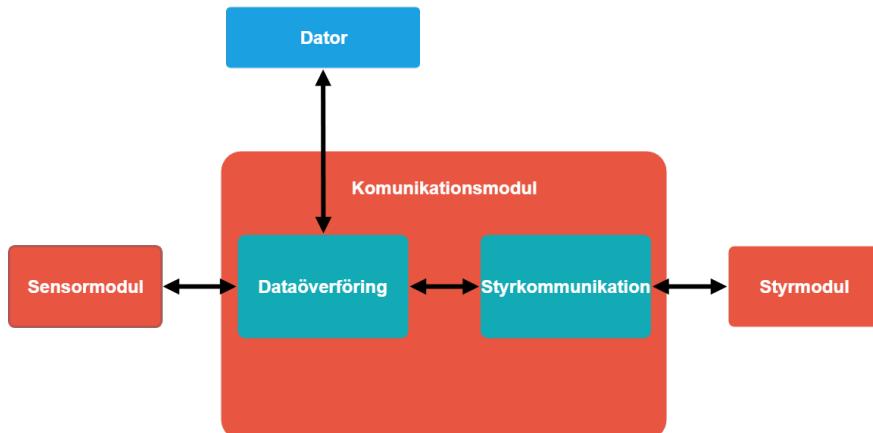
Krav	Ändring	Beskrivning	Prioritet
1	Original	Bilen ska kunna köra autonomt. (se krav 55-62)	1
2	Original	Bilen ska kunna köra semi-autonomt. (se krav 61-70)	1
3	Original	Bilen ska kunna köra manuellt. (se krav 71)	1
4	Original	Bilen ska ha en kamera.	1
5	Original	Bilen ska vara modulärt uppbyggd.	1
6	Original	Varje modul ska innehålla en processor.	1
7	Original	Bilen ska innehålla en styrmodul.	1
8	Original	Bilen ska innehålla en kommunikationsmodul.	1
9	Original	Bilen ska innehålla en sensormodul.	1
10	Original	Bilen ska ha en inbyggd manuell strömbrytare.	1
11	Original	Bilen ska kunna matas med en karta av ett vägnät.	1
12	Original	Bilen ska köra på ett projicerat vägnät enligt appendix A.	1
13	Original	Bilen ska kunna köra till olika destinationer enligt appendix A på kortast möjliga väg.	1
14	Original	Under körning ska relevant data visas på en datorskärm.	1
15	Original	Relevant data ska kunna sparas på ett format så att man kan plotta valda signaler som funktion av tiden.	1
16	Original	Kamerabilder från en körning ska kunna sparas.	1
17	Original	Kommunikation mellan bilen och persondatorn ska vara trådlös	1



UNTER

3 KOMMUNIKATIONSMODUL

Kommunikationsmodulen ansvarar för kommunikationen mellan användaren och bilen. Bilen ska med hjälp av denna modul kunna tolka kommandon som kommer från en bärbar dator samt skicka information om t.ex. den senaste körningen tillbaka till datorn. Daten ska kunna sparas och plottas på datorn på ett användarvänligt sätt. För skiss över kommunikationsmodulen se figur 2.



Figur 2: Kommunikationsmodulen

3.1 Inledande beskrivning

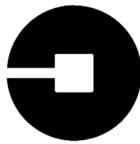
Användaren kommer att presenteras ett grafiskt gränssnitt som låter användaren köra bilen trådlöst via en bärbar dator. Detta gränssnitt ska även visa all relevant data (enligt tabell 5) på ett tydligt sätt.

3.2 Gränssnitt

Kommunikationsmodulen ska kunna skicka och ta emot data ifrån använderen (körförskjutning, kördata), styrmoden (initialisering av parametrar, kördata) och sensormoden (inställningar, sensordata).

Tabell 3: Kravlista för gränssnitt kommunikationsmodul

Krav	Ändring	Beskrivning	Prioritet
18	Original	Kommunikationsmodulen ska kunna ta emot och skicka data till styrmoden.	1
19	Original	Kommunikationsmodulen ska kunna ta emot och skicka data till sensormoden.	1
20	Original	Kommunikationsmodulen ska kunna ta emot och skicka data till en bärbar dator.	1



UNTER

3.3 Designkrav

I tabell 4 listas designkraven på kommunikationsmodulen.

Tabell 4: Kravlista för designkrav kommunikationsmodulen

Krav	Ändring	Beskrivning	Prioritet
21	Original	Kommunikationsmodulen ska få plats på bilen.	1
22	Original	Kommunikationsmodulen ska ha en processor.	1

3.4 Funktionella krav

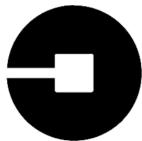
Tabell 5 listar de funktionella kraven för kommunikationsmodulen.

Tabell 5: Kravlista för funktionella krav kommunikationsmodul

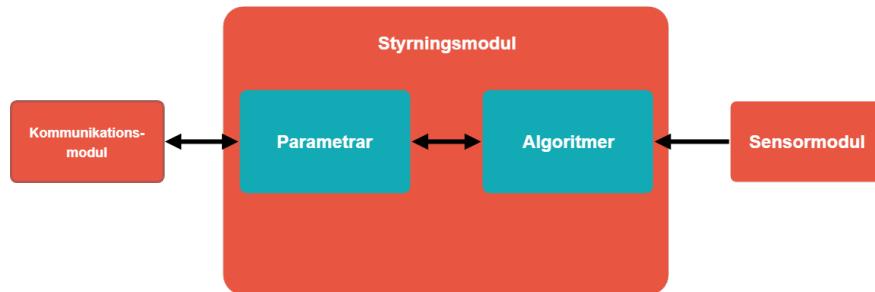
Krav	Ändring	Beskrivning	Prioritet
23	Original	Kommunikationsmodulen ska kunna ta emot körkommandon från en bärbar dator (se prestandakrav, tabell 15).	1
24	Original	Styrkommandon ska kunna skickas till styrmodulen. (se prestandakrav, tabell 15).	1
25	Original	Bilen ska överföra information om körtid till en bärbar dator under körning.	1
26	Original	Bilen ska överföra information om körsträcka till en bärbar dator under körning.	1
27	Original	Bilen ska överföra information om hastighet till en bärbar dator under körning.	1
28	Original	Bilen ska överföra information om detektion och avstånd till hinder till en bärbar dator under körning.	1
29	Original	Bilen ska överföra information om gaspådrag till en bärbar dator under körning.	1
30	Orginal	Bilen ska överföra information om styrutslag till en bärbar dator under körning.	1

4 STYRMODULEN

Styrmodulen ansvarar för all styrning av bilen. Styrmodulen har all funktionalitet för att bilen ska kunna köra framåt och bakåt i hjulens riktning, att bilen kan stanna samt att bilen kan svänga för att ändra körriktningen. Styrmodulen ansvarar även för styrregleringen så att bilen ej slingrar sig fram vid körning. För komplett skiss över styrmodulen se figur 3.



UNTER



Figur 3: Styrmodul

4.1 Inledande beskrivning av styrmodulen

Styrmodulen tar emot körkommandon (gaspådrag, styrutslag) till en mikrodator från kommunikationsmodulen. Styrmodulens uppgift är att tolka dessa kommandon och kontrollera bilen på angivet sätt.

4.2 Externa gränssnitt

I tabell 6 listas de krav styrmodulen har gentemot systemets andra moduler.

Tabell 6: Kravlista för gränssnitt styrmodul

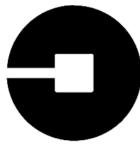
Krav	Ändring	Beskrivning	Prioritet
31	Original	Styrmodulen ska kunna ta emot data från och skicka data till kommunikationsmodulen.	1
32	Original	Styrmodulen ska kunna ta emot data från sensormodulen.	1

4.3 Designkrav

Tabell 7 listar designkraven för styrmodulen.

Tabell 7: Kravlista för designkrav styrmodul

Krav	Ändring	Beskrivning	Prioritet
33	Original	Styrmodulen ska få plats på bilen.	1
34	Original	Styrmodulen ska vara enkel att byta ut.	1



UNTER

4.4 Funktionella krav

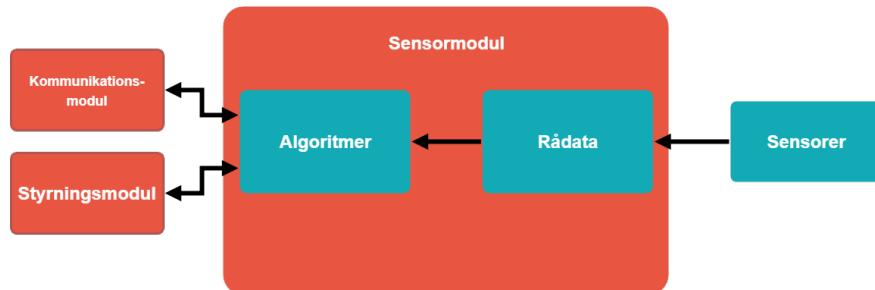
De funktionella kraven i tabell 8 listar kraven på funktionalitet hos styrmodulen.

Tabell 8: Kravlista för funktionella krav styrmodul

Krav	Ändring	Beskrivning	Prioritet
35	Original	Styrmodulen ska reagera på körkommandon från kommunikationsmodulen (se prestandakrav, tabell 15).	1
36	Original	Parametrar för styralgoritmen ska kunna initieras trådlöst.	1
37	Original	Styrmodulen ska ha styrreglering så bilen ej slingrar sig fram.	1

5 SENSORMODUL

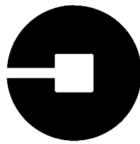
Sensormodulen ansvarar för alla sensorer samt kameran på bilen och den sensordata dessa sensorer genererar. Sensormodulen ansvarar för att översätta insamlad data från sensorer till användbara storheter för bilens andra system. För komplett skiss över sensormodulen se figur 4.



Figur 4: Sensormodul

5.1 Inledande beskrivning av sensormodulen

Sensormodulen utgörs av ett antal sensorer samt en kamera som är kopplad till en mikroprocessor. Modulen omvandlar insamlad data till användbara storheter, exempelvis meter och meter per sekund, och skickar denna modifierade data vidare till bilens andra delsystem där det används som beslutsunderlag.



UNTER

5.2 Externa gränssnitt

I tabell 9 listas de krav sensormodulen har gentemot systemets andra moduler.

Tabell 9: Kravlista för gränssnitt sensormodul

Krav	Ändring	Beskrivning	Prioritet
38	Original	Sensormodulen ska kunna ta emot data ifrån och skicka data till kommunikationsmodulen.	1
39	Original	Sensormodulen ska kunna skicka data till styrmodulen.	1

5.3 Designkrav

Tabell 10 listar designkraven för sensormodulen.

Tabell 10: Kravlista för designkrav sensormodul

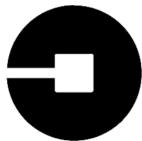
Krav	Ändring	Beskrivning	Prioritet
40	Original	Alla sensorer ska vara kopplade till sensormodulen.	1
41	Original	Sensormodulen ska få plats på bilen.	1

5.4 Funktionella krav

I tabell 11 beskriver de krav på funktionalitet hos sensormodulen.

Tabell 11: Kravlista för funktionella krav sensormodul

Krav	Ändring	Beskrivning	Prioritet
42	Utgick 2019-05-08	Sensormodulen ska kunna avgöra om bilen är utanför banan.	1
43	Original	Sensormodulen ska kunna beräkna hastighet.	1
44	Original	Sensormodulen ska kunna beräkna avstånd till hinder.	1



UNTER

6 PERSONDATOR

Persondatorn har till uppgift att kommunicera med delsystemen i bilen. Användaren matar in kommandon i persondatorn som sedan trådlöst överförs till bilen.

6.1 Inledande beskrivning av persondatorn

Via persondatorn kan användaren, via kommandon i ett konstruerat användargränssnitt, kommunicera med bilen. Kommunikationen ska ske trådlöst till bilen och den ska tolka kommandona och utföra användarens angivna uppdrag. Bilen ska även kunna kommunicera med persondatorn och därmed, på persondatorns skärm, kunna rita upp en karta över vägnätet samt presentera grafer.

6.2 Externa gränssnitt

Tabell 12 listar de krav persondatorn har gentemot de andra modulerna.

Tabell 12: Kravlista för gränssnitt persondator

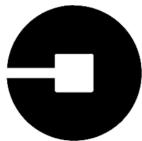
Krav	Ändring	Beskrivning	Prioritet
45	Original	Persondatorn ska kunna ta emot data ifrån och skicka data till kommunikationsmodulen.	1

6.3 Designkrav

Tabell 13 beskriver de desginkrav hos persondatorn för att kunna kommunicera med bilen.

Tabell 13: Kravlista för designkrav persondator

Krav	Ändring	Beskrivning	Prioritet
46	Utgick 2019-05-08	Persondatorn ska ha ett användarvänligt gränssnitt.	2
47	Original	Persondatorn ska kunna plotta körtid, körsträcka, hastighet, detektion och avstånd till hinder föröver, gaspådrag och styrutslag som en funktion av tiden för en given körning.	1
48	Original	Persondatorn ska i autonomt läge visa planerad rutt till nästa destination.	1



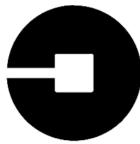
UNTER

6.4 Funktionella krav

Tabel 14 beskriver den funktionalitet som krävs för persondatorn.

Tabell 14: Kravlista för funktionella krav persondator

Krav	Ändring	Beskrivning	Prioritet
49	Original	Persondatorn ska kunna ta emot mätvärden från bilen och lagra dessa.	1
50	Original	Persondatorn ska kunna skicka styrkommandon till kommunikationsmodulen för autonom körning.	1
51	Original	Persondator ska kunna skicka kördirektiv (se prestandakrav, tabell 15) till kommunikationsmodulen för semi-autonom körning.	1
52	Original	Persondatorn ska kunna köra bilen manuellt.	1



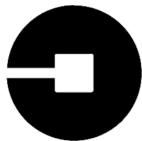
UNTER

7 PRESTANDAKRAV

I tabell 15 nedan redogörs prestandakraven för bilen. Prestandakraven är de krav som utgör hur väl bilen ska fungera och vad dess huvudsakliga syfte är.

Tabell 15: Prestandakrav

Krav	Ändring	Beskrivning	Prioritet
53	Original	Bilen ska köra på vägen utan att "slingra" sig (verifieras genom dokumenterade testkörningar).	1
54	Original	Under körning skickar bilen fortlöpande positionsdata till datorn som visar positionen på en karta.	3
55	Original	Autonom mod: bilen ska kunna stanna framför ett hinder på vägen.	1
56	Original	Autonom mod: bilen ska kunna byta fil och köra runt ett hinder på vägen.	3
57	Original	Autonom mod: bilen ska kunna stanna framför en stopplinje.	1
58	Original	Autonom mod: bilen ska kunna köra in och parkera i en parkeringsficka.	3
59	Original	Autonom mod: planerad rutt till nästa destination ska visas på en bärbar dator.	1
60	Original	Autonom mod: bilen ska kunna ta emot och genomföra ett autonomt uppdrag (beskrivet i appendix A).	1
61	Original	Autonom och semi-autonom mod: aktuellt vägsegment och lateral position ska visas på en bärbar dator.	1
62	Original	Autonom och semi-autonom mod: Bilen ska autonomt kunna variera hastigheten under körning.	3
63	Original	Semi-autonom mod: bilen ska kunna stanna framför ett hinder på vägen.	1
64	Original	Semi-autonom mod: bilen ska kunna byta fil och köra runt ett hinder på vägen.	3
65	Original	Semi-autonom mod: bilen ska kunna stanna framför en stopplinje.	1
66	Original	Semi-autonom mod: bilen ska kunna köra in och parkera i en parkeringsficka.	3
67	Original	Semi-autonom mod: planerad rutt till nästa destination ska visas på en bärbar dator.	1
68	Original	Semi-autonom mod: bilen ska kunna ta emot och genomföra ett autonomt uppdrag (beskrivet i appendix A).	1
69	Utgick 2019-05-08	Semi-autonom mod: aktuellt vägsegment och lateral position ska visas på en bärbar dator.	1
70	Original	Semi-autonom mod: Användaren ska kunna skicka vägvisningskommandon, t.ex. följ vägen fram till nästa rondell, ta andra avfarten, stanna.	1
71	Original	Manuell mod: bilen ska reagera på följande manuella kommandon: gaspådrag inklusive fram/back samt styrutslag.	1



UNTER

8 KRAV PÅ VIDAREUTVECKLING

För framtida utveckling ska bilen kunna uppgraderas till att kunna köra i ett trafikerat vägnät. Bilen ska även vid framtida vidareutveckling kunna köra på mera varierat vägunderlag, exempelvis backar och regniga vägar.

9 TILLFÖRLITLIGHET

I tabell 16 presenteras kraven på bilens tillförlitlighet.

Tabell 16: Tillförlitlighet

Krav	Ändring	Beskrivning	Prioritet
72	Original	Bilen ska klara av sitt uppdrag tre av fyra gånger.	1

10 EKONOMI

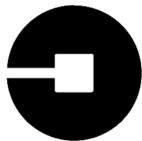
I tabell 17 presenteras gruppens ekonomiska krav som framförallt kommer att begränsas av antal arbetstimmar.

Tabell 17: Ekonomikrav

Krav	Ändring	Beskrivning	Prioritet
73	Original	Projektet ska ta 230 timmar per person att slutföra efter BP2.	1

11 KRAV PÅ SÄKERHET

Bilen ska konstrueras på ett sådant sätt att den inte är en fara för sin omgivning. Till exempel ska det inte finnas en risk att användaren utsätts för höga spänningar eller skadligt stor värmeutveckling.



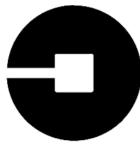
UNTER

12 LEVERANSKRAV OCH DELLEVERANSER

I tabell 18 redogörs leveranskraven för projektet.

Tabell 18: Leveranskrav och delleveranser

Krav	Ändring	Beskrivning	Prioritet
74	Original	Kravspecifikationen ska vara klar, reviderad och inlämnad senast 2019-02-05 kl 16:00.	1
75	Original	Första versionen av projektplan, tidplan och systemskiss ska vara inlämnade till beställaren senast 2019-02-15 kl 16:00.	1
76	Original	Slutgiltig version av projektplan, tidplan och systemskiss ska vara inlämnade till beställaren senast 2019-02-22 kl 16:00.	1
77	Original	Första version av gruppens förstudier ska vara skickad till handledare och beställare senast 2019-03-04 kl 16:00.	1
78	Original	Första versionen av designspecifikationen ska vara inlämnad till handledaren senast 2019-03-08 kl 16:00.	1
79	Original	Designspecifikationen ska vara inlämnad och godkänd av handledaren senast 2019-03-15.	1
80	Original	Version 1.0 av gruppens förstudier ska vara skickade till handledare och beställare senast 2019-03-27 kl 16:00.	1
81	Original	Dåvarande design ska vara presenterad för och godkänd av handledaren senast 2019-04-09.	1
82	Original	Demonstration av baskrav, presentation av statusrapporten och diskussion om projektets slutfas, planering och slutkrav ska ske senast 2019-05-07.	1
83	Original	Kappan version 1.0 ska vara inlämnad till beställaren senast 2019-05-21.	1
84	Original	Verifiering av slutkraven ska ske mellan 2019-05-20 och 2019-05-22.	1
85	Original	Teknisk dokumentation (version 1.0) och användarhandledning (version 1.0) ska vara inlämnade till beställaren senast 2019-05-22.	1
86	Original	Tävlingar och demonstrationer ska ske 2019-06-04.	1
87	Original	Efterstudien ska vara inlämnad till beställaren senast 2019-06-10.	1
88	Original	All utrustning och nycklar ska vara återlämnade senast 2019-06-10.	1
89	Original	En tidsrapport som dokumenterar tiden för skrivning av kravspecifikation ska skickas till beställaren senast 16:00 2019-02-06.	1
90	Original	En tidsrapport som dokumenterar tiden för skrivning av systemskiss och projektplan ska skickas till beställaren senast 16:00 2019-02-25.	1
91	Original	En tidsrapport tillsammans med en uppdaterad tidplan ska skickas till beställare senast kl 16.00 vid följande datum: 2019-03-11, 2019-03-25, 2019-04-01, 2019-04-08, 2019-04-15, 2019-04-29, 2019-05-06, 2019-05-13, 2019-05-20, 2019-05-27, 2019-06-03, 2019-06-10.	1

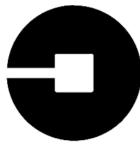
**UNTER**

13 DOKUMENTATION

Tabell 19 listar de dokument som skall produceras.

Tabell 19: Dokument som skall produceras

Dokument	Språk	Syfte	Målgrupp	Format
Kravspecifikation	Svenska	Entydig bestämning av kraven som ställs på projektet och den färdiga produkten. Ligger som underlag för bilens design och implementation.	Projektgrupp och Beställare	pdf
Systemskiss	Svenska	En översiktlig beskrivning om hur bilen ska konstrueras, med grund i kravspecifikationen. Används för att avgöra vilka moduler som bilen innehåller och hur konstruktionen av dessa delas in i arbetsblock.	Projektgrupp och handledare	pdf
Projektplan	Svenska	Bestämning av projektet utförande och vilka villkor som gäller projektmedlemmar sinsemellan.	Projektgrupp	pdf
Tidplan	Svenska	Används för att spåra gruppmedlemmars tidsåtgång samt för att dela upp arbetet i tidsblock.	Projektgrupp	xls
Designspecifikation	Svenska	Förfining av systemskiss som i mer detalj beskriver hur bilen ska konstrueras. Inkluderar kretsschema och flödesschema.	Projektgrupp och handledare	pdf
Förstudie	Svenska	Dokument som introducerar hur man skriver på ett tekniskt och vetenskapligt sätt.	Projektgrupp	pdf
Mötesprotokoll	Svenska	Dokument som ligger till underlag för uppföljning av projektet.	Projektgrupp och Beställare	doc
Teknisk dokumentation	Svenska	Dokument som beskriver produktens konstruktion.	Beställare	pdf
Användarhandledning	Svenska	Dokument som beskriver för användaren hur produkten ska användas.	Beställare	pdf
Efterstudie	Svenska	Sammanfattande dokument som sammanställer projektgruppens erfarenheter av arbetssätt, samarbete och användandet av projektmodellen.	Projektgrupp och beställare	pdf



UNTER

14 UTBILDNING

För att gruppen ska kunna leverera produkten till beställaren så krävs det att gruppen genomgår ett antal utbildningar. Dessa kommer att till största delen bestå av laborationer men även moment som etik och dokumentskrivning kommer att ingå. Kraven för gruppen presenteras i tabell 20.

Tabell 20: Utbildning

Krav	Ändring	Beskrivning	Prioritet
92	Original	Gruppen ska genomföra en laboration med fokus på AVR.	1
93	Original	Gruppen ska genomföra en laboration med fokus på mätteknik.	1
94	Original	Gruppen ska genomföra en föreläsningsserie med fokus på etik inom ingenjörsmässiga problem.	1
95	Original	Gruppen ska genomföra två etikseminarier relaterade till föreläsningarna i krav 94.	1
96	Original	Gruppen ska genomföra en föreläsningsserie med fokus på hur man skriver relevanta dokument för projektet. Exempelvis kravspecifikation, systemskiss, projektplan, m.m.	1

15 KVALITETSKRAV

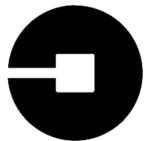
I tabell 21 beskrivs projektets kvalitetskrav.

Tabell 21: Kvalitetskrav

Krav	Ändring	Beskrivning	Prioritet
97	Original	Projektet ska bedrivas enligt LIPS-modellen.	1
98	Original	Projektets dokument ska utgå, då det är möjligt, utifrån LIPS-mallar.	1
99	Original	Projektet ska bedrivas med Svenska Ingenjörers hederskodex i åtanke.	1

16 UNDERHÅLLSBARHET

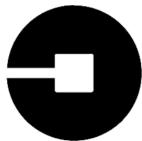
Bilen ska konstrueras på ett sätt som gör att det blir enkelt att underhålla den. Till exempel genom att bilden är modulärt uppbyggd för att enkelt kunna byta ut eller uppdatera olika delsystem utan att förlora funktionalitet. Genom rutinmässiga kontroller säkerställs att bilden alltid är hel och kan prestera med bästa möjliga prestanda.



UNTER

REFERENSER

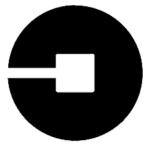
- [1] S. Språknämnden, *Svenska skrivregler*, 2nd ed. Liber AB, Stockholm, 2000, ISBN47-04974-X.



UNTER

A APPENDIX

I det bifogade appendix A finns projektets gemensamma dokument om banspecifikation och tävlingsregler.



UNTER

B BANSPECIFIKATION OCH TÄVLINGSREGLER

Appendix B

Leah Engman, Anton Revin och Nicholas Sepp

4 juni 2019

Version 1.0

Status

Granskad	LE, AR, NS	4 juni 2019
Godkänd	Anders Nilsson	4 juni 2019

Dokumentförfattare

Namn	Grupp	Telefon	E-post
Leah Engman	7	+46 (0)70-4398877	leaen879@student.liu.se
Anton Revin	8	+46 (0)72-1978761	antre986@student.liu.se
Nicholas Sepp	9	+46 (0)73-9790777	nicse725@student.liu.se

Dokumenthistorik

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	4 februari 2019	Första utkast	LE, AR, NS	LE, AR, NS
0.2	5 februari 2019	Andra utkast	LE, AR, NS	LE, AR, NS
1.0	4 juni 2019	Godkänd upplaga	LE, AR, NS	LE, AR, NS

INNEHÅLL

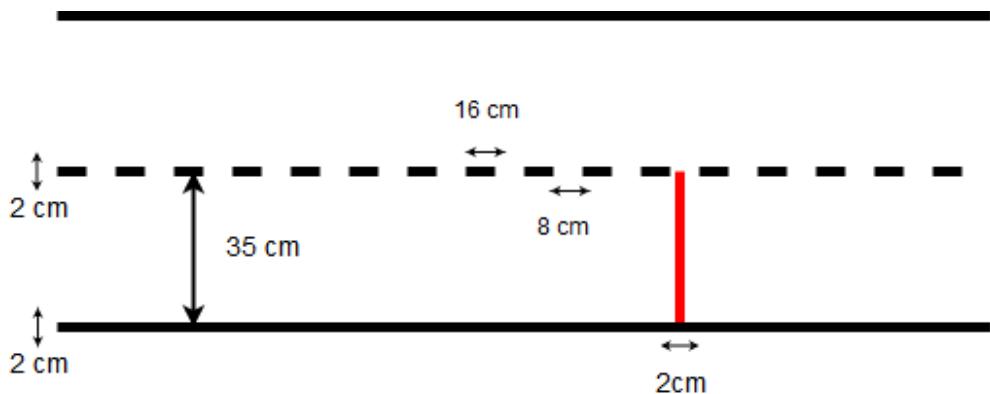
1	Banspecifikation	1
1.1	Vägar	1
1.2	Hinder	1
1.3	Parkeringsfickor	2
1.4	Rondeller	3
2	Tävlingsregler	4
2.1	Regler för heat	4
2.2	Diskvalificering	4
2.3	Poängsystem	5
2.4	Begränsningar	5
2.5	Domare	5

1 BANSPECIFIKATION

Banans utformning ska följa ett antal bestämmelser på bland annat utseende och dimensioner. Det ska finnas tre olika bansegment: raksträckor, kurvor och rondeller. Banan får totalt innefatta en area på *10 x 5 meter*.

1.1 Vägar

Banans vägar består av två filer. En fil är *35 cm* bred. De två filerna separeras av en svart, streckad linje där varje streck är *2 cm* bred, *16 cm* lång och separeras av *8 cm* tomrum. Vägen begränsas av en heldragen svart linje på vardera sida som är *2 cm* bred. En stopplinje är en heldragen, *2 cm* bred rödfärgad linje som dras tvärs över en fil.



Figur 1: Ett vägsegment med dimensioner

I ett vägnät kan det förekomma raksträckor som kan variera i längd från *1 m* till *5 m*. Om en väg svänger är minsta tillåtna kurvradien *87,5 cm* (samma som rondellerna vilka definieras i avsnitt 1.4).

1.2 Hinder

Samtliga hinder som förekommer på banan ska följa följande förutbestämda mått:

- Hindret ska täcka $\frac{3}{4}$ av filen i bredd, placerad centrerad på filen.
- Hindret ska vara en billängd lång, alltså *35 cm*.
- Hindret ska vara minst *15 cm* högt.

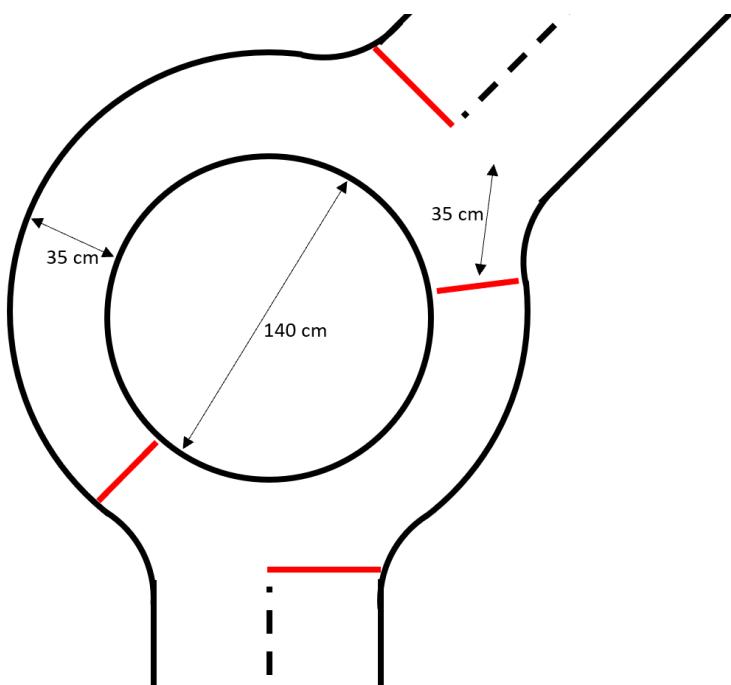
1.3 Parkeringsfickor

En parkeringsficka ska placeras vid samtliga destinationer, dvs strax efter en stopplinje. Alla parkeringsfickorna ska vara av samma storlek och utseende och uppfylla följande krav:

- Parkeringsfickan ska vara 1,5 billängd lång.
- Parkeringsfickan ska vara lika bred som övriga filer, alltså *35 cm*.
- Parkeringsfickan ska ha en yttre heldragen markering samt en streckad avgränsning in mot vägen.
- Parkeringsfickan ska vara placerad till höger.
- Parkeringsfickan ska likt rondellerna ha mjuka (rundade) infarter och utfarter.

1.4 Rondeller

Banan innehåller en uppsättning rondeller som sammankopplar dess vägar. I ett vägnät får det förekomma två till fyra rondeller. Dessa är alla av samma storlek och konstrueras enligt dimensionerna vilka beskrivs i Figur 2. Samtliga linjer följer specifikationerna under avsnitt 1.1 och har alltså bredden 2 cm . Alla rondeller har samma storlek men kan ha ett varierande antal förbindande vägar vilka markeras med en röd stopplinje (enligt avsnitt 1.1) en billängd ifrån den anslutande vägens start. Det ska existera minst två och maximalt fyra av/på-farter till varje rondell. Samtliga på- samt avfarter har rundade kanter mot rondellens fil. Rondellerna består av en inneliggande heldragen ring med innerdiameter 140 cm följt av en fil med samma dimension som de anslutande vägarna (definierad i avsnitt 1.1): 35 cm bred.



Figur 2: Dimensioner på banans rondeller

2 TÄVLINGSREGLER

Under tävlingen kör deltagande grupper sammanlagt två heat: ett heat var på två olika banor. Banornas och köruppdragens utformning bestäms av projektbeställaren utifrån de regler som definierats under avsnitt 1 och de deltagande grupperna får inte ta del av dem i förhand. Efter att alla grupper kört sitt heat bestäms en rangordning från första plats till tredje plats (alternativt kan ett heat bli diskvalificerat, se underrubriken 2.2 Diskvalificering). Rangordningen bestäms av vilken grupp som kör heatet på kortast tid och varje rang ger gruppen en bestämd poäng (se underrubriken 2.3 Poängsystem). Den grupp med minst antal poäng efter att alla heat har körts vinner hela tävlingen. Om två (eller fler) grupper har samma antal poäng efter att båda heat har körts jämförs gruppernas sammanlagda heat-tider. Om en av grupperna vars tider jämförs har ett diskvalificerat heat vinner den grupp som inte har något diskvalificerat heat. Om båda grupperna har diskvalificerade heat samt besitter samma antal poäng så delar grupperna placering.

2.1 Regler för heat

Tävlingen består av två heat. Det första heatet körs på en bana med ett köruppdrag och det andra heatet körs på en annan bana med ett annat köruppdrag. Varje deltagande lag kör efter samma köruppdrag i varje respektive heat. Ett köruppdrag får innehålla 3 till 5 destinationer.

Vid introduktion av ny bana får grupperna mata in bankartan till deras bilar. Grupperna får även mäta avstånd på banan och mata in den data de känner att de behöver till deras bilar. Högertrafik gäller, t.ex. får inga olagliga U-svängar göras. Inför början av ett heat ska den aktuella bilen placeras på heatets startpunkt och så fort köruppdraget har skickats över till bilen påbörjas tidtagning. Tidtagning avslutas så fort den aktuella bilen har stannat vid köruppdragets slutdestination.

2.2 Diskvalificering

Diskvalificering av ett heat sker vid vissa specificerade händelser, som följer:

- Om bilen kör av banan helt (enligt bedömning av tävlingens domare).
- Om bilen stannar före en stopplinje på ett avstånd större än en halv billängd alternativt om bilen kör över en stopplinje helt och hållt då den förväntas stanna vid den (destination i köruppdraget).
- Om bilen kör olagligt, t.ex. om bilen gör en olaglig U-sväng eller passerar vägens mitträcke i, av tävlingens domare, bestämd grad.

2.3 Poängsystem

Efter varje heat bestäms en rangordning baserat på vilken grupp som har kortast tid och varje placering ger gruppen en viss poäng. Dessa poäng ser ut som följande:

- Första plats: 1 poäng
- Andra plats: 2 poäng
- Tredje plats: 3 poäng
- Diskvalificerad: 5 poäng

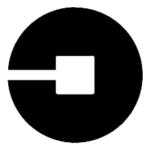
2.4 Begränsningar

Utformningen av banorna som grupperna ska tävla på ska vara begränsad. Dessa begränsningar är som följande:

- Tävlingsbanan ska inte ha några hinder.
- Tävlingsbanan ska inte ha några parkeringsfickor.
- Inget av köruppdragets destinationer ska finnas i eller inom 2 billängder av en rondell.

2.5 Domare

Beställaren Anders Nilsson agerar som tävlingens enväldiga domare.



UNTER

C SYSTEMSKISS



UNTER

Autonom taxibil

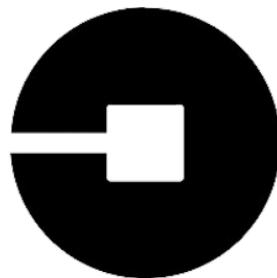
4 juni 2019

Systemskiss

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl

4 juni 2019

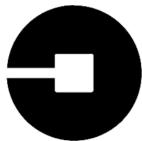
Version 1.0



UNTER

Status

Granskad	OM, EM, NS	2019-02-22
Godkänd	Anders Nilsson	2019-02-26

**UNTER****Projektidentitet**Grupp E-post: jonca673@student.liu.se

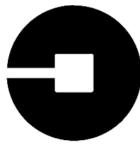
Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

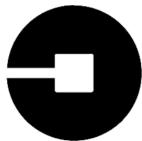
Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

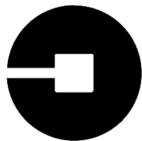
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Designansvarig (DES)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se

**UNTER****INNEHÅLL**

1	Inledning	1
2	Översikt av systemet	1
3	Kommunikationsmodul	3
3.1	Inledande beskrivning av kommunikationsmodulen	3
3.2	Trådlös kommunikation	4
3.3	Bussar	4
3.4	Brister	4
4	Styrmodul	5
4.1	Inledande beskrivning av styrmodulen	5
4.2	Motor	5
4.3	Styrservo	5
4.4	Styrreglering	5
4.5	Brister	5
5	Sensormodul	6
5.1	Inledande beskrivning av sensormodulen	6
5.2	Kamera	6
5.3	Avståndssensor	7
5.4	Vinkelhastighetssensor	7
5.5	Brister	7
6	Extern dator	7
6.1	Inledande beskrivning av extern dator	8
6.2	Användargränssnitt	8
6.3	Brister	8

**UNTER****DOKUMENTHISTORIK**

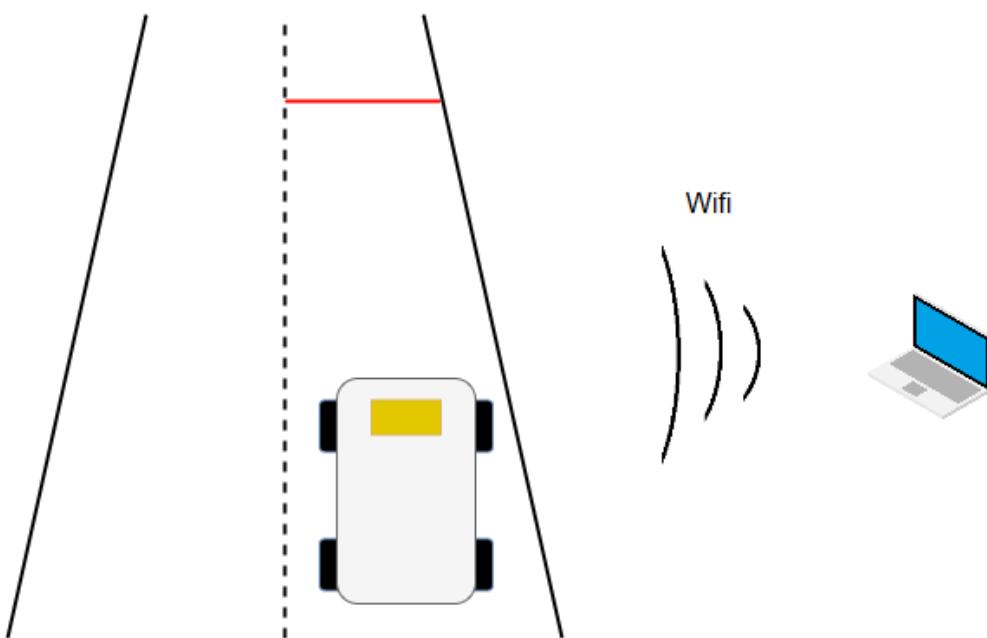
Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2019-02-11	Första utkast	Grupp 9	OM, EM, NS
1.0	2019-02-26	Andra utkast	Grupp 9	OM, EM, NS



UNTER

1 INLEDNING

I detta projektarbete ska en autonom taxibil konstrueras. Bilen ska autonomt kunna följa ett köruppdrag på ett vägnät som projiceras i Visionen. Hela systemet, taxibilen med tillhörande persondator, i dess omgivning illustreras i figur 1. Detta dokument ska ge ett underlag för hur produkten ska konstrueras i stora drag. Tidigare beskrivna modulers funktionalitet detaljeras ytterligare, detsamma med kommunikationen mellan dem. Även bilens kommunikation med den externa datorn detaljeras, bland annat hur data från bilen ska visas i datorns användargränssnitt. Även bilens chassis och hårdvarans position redogörs för översiktligt, se figur 2.

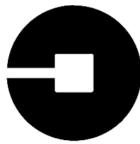


Figur 1: Systemet i sin omgivning

2 ÖVERSIKT AV SYSTEMET

Nedan redogörs för en översikt av systemet, bland annat hur delsystemen sitter ihop och gränssnitten mellan dem. Hela *kommunikationsmodulen* implementeras som en process i en mikrodator (en Raspberry Pi). Kommunikationen mellan kommunikationsmodulen och den *externa persondatorn* utnyttjar mikrodatorns inkopplade WiFi-adAPTER för att överföra information trådlöst; t.ex. ska bilen skicka data som hastighet och körtid till datorn för att visas och till bilen ska datorn skicka t.ex. körförslag (i det semi-autonoma läget). Bildbehandlingen av kamerans bilddata görs externt i en process i Raspberry Pi:n. En modul är alltså inte bunden till en enskild processor, den kan ta hjälp av extern processorkraft för att göra sina beräkningar.

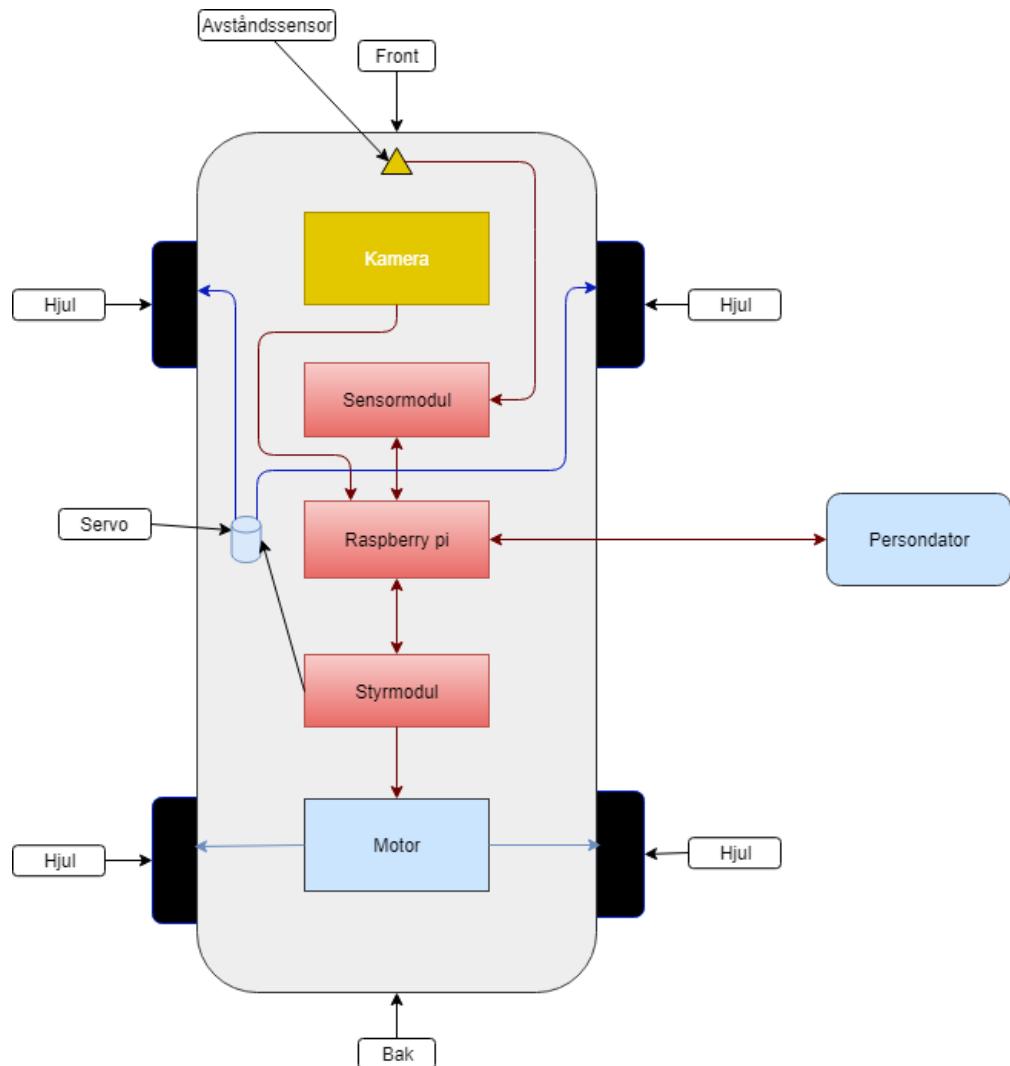
Sensormodulen behöver omvandla mätdata till praktisk användbar data för de andra modulerna. *Styrmodulen* innehåller styr- och regleralgoritmer och ser till att omvandla högnivåkommandon från kommunikationsmodulen till styrsignaler



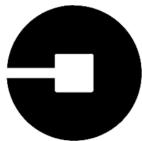
UNTER

åt bilens motor, styr servo, etc. Både sensormodulen och styrmodulen kommer att använda en AVR-processor från Atmel för att göra sina beräkningar.

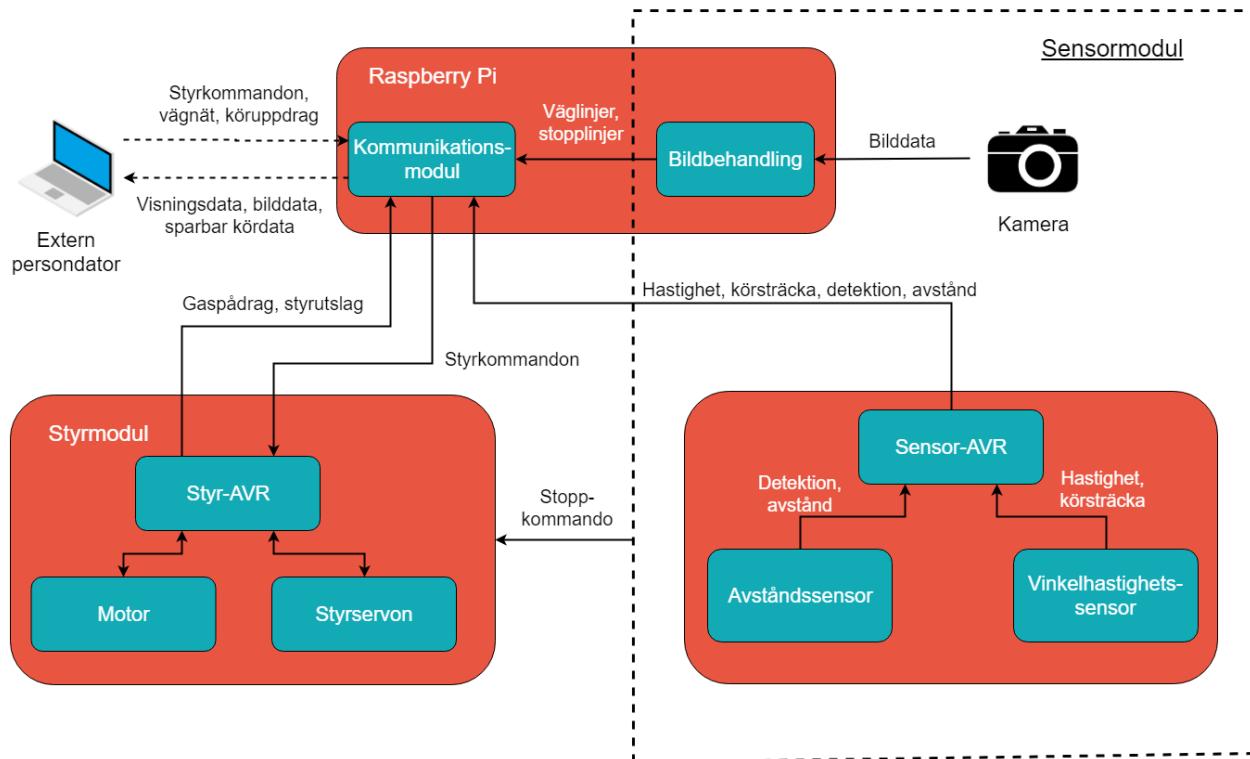
Nedan i figur 2 visas en överblick av bilens chassi och hårdvaran ovanifrån. I figur 3 visas en överblick av dataflödet mellan delsystemen.



Figur 2: Överblick av bilen och dess hårdvara sett ovanifrån.



UNTER



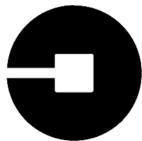
Figur 3: Överblick av bilens delsystem.

3 KOMMUNIKATIONSMODUL

I detta kapitel beskrivs kommunikationsmodulen i detalj. För referens, se figur 4.

3.1 Inledande beskrivning av kommunikationsmodulen

Kommunikationsmodulens uppgift är att sköta kommunikation mellan styrmoden, sensormoden och persondatorn. Persondatorn behöver information från körningen som finns dels i styrmoden och dels i sensormoden. Denna överföring sker i två steg, intern överföring till kommunikationsmodulen och extern överföring till datorn (trådlös). Bilens interna dataöverföring mellan modulerna sker med bussar. För att möjliggöra den trådlösa kommunikationen används WiFi-adaptern som är kopplat till en Raspberry Pi. Denna processor används också för alla beräkningar som är relaterade till kommunikationsmodulen.



UNTER

3.2 Trådlös kommunikation

Projektgruppen har valt att använda sig av WiFi för den trådlösa kommunikationen mellan persondatorn och kommunikationsmodulen. Detta för att det är lätt att koppla en WiFi-adapter till en Raspberry Pi som sitter på bilen. Den trådlösa kommunikationen ske i båda riktningar där kommunikationsmodulen skickar data från bilens moduler till persondatorn och persondatorn skickar kommandon till kommunikationsmodulen.

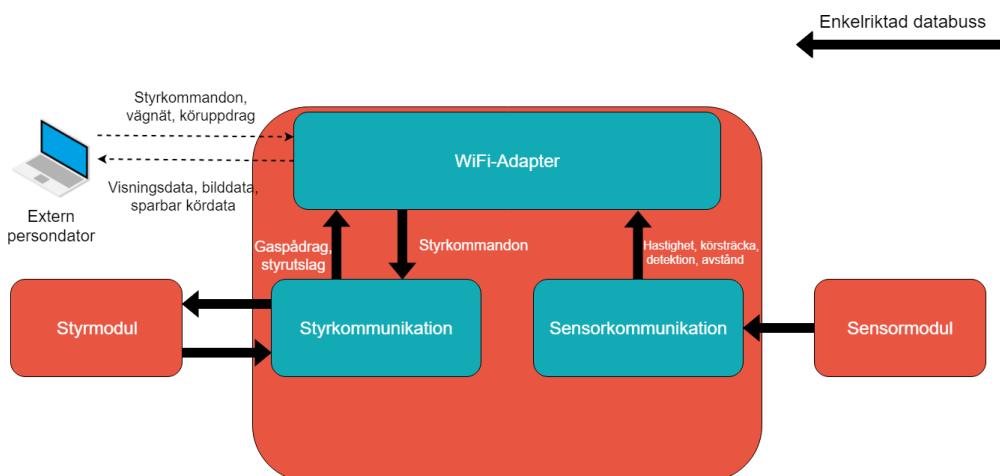
3.3 Bussar

Kommunikationen mellan kommunikationsmodulen och bilens andra moduler kommer att ske internt via seriella bussar där kommunikationsmodulen agerar master och styr- samt sensormodulen agerar slavar. Det vill säga att kommunikationsmodulen kommer att hantera den generella kommunikationen mellan bilens moduler.

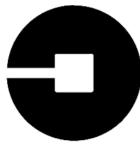
3.4 Brister

WiFi-adaptern på Raspberry Pi:n har ej obegränsad räckvid. Om bilen kör för långt ifrån persondatorn kan bilens kommunikationsmodul tappa kontakten med persondatorn och data kan då ej skickas till datorn och nya kommandon kan ej tas emot.

Master-slave förhållandet för den interna bussen innebär att kommunikationsmodulen (master) kan kommunicera med både styr- och sensormodulen (slavar) men slavarna kan ej tala med varandra. Gruppen tänker implementera en annan separat buss mellan styr- och sensormodulen så att de oberoende kan kommunicera för att motverka denna brist.



Figur 4: Systemskiss av kommunikationsmodulen



UNTER

4 STYRMODUL

I detta kapitel beskrivs styrmodulen i detalj. För referens, se figur 5.

4.1 Inledande beskrivning av styrmodulen

Styrmodulens uppgift är att förflytta och styra bilen. Bilens motor behöver en tillförd spänning för att driva bilen framåt och styrservot behöver ställas om för att bilen ska kunna svänga i olika riktningar. Styrmodulen ansvara även för bilens styrreglering så att bilen kan färdas på vägen utan att köra utanför.

4.2 Motor

Bilen kommer att ha en motor som när tillförd en spänning kan föra bilen framåt. För att göra detta kommer motorn behöva en tillförd spänning som kommer från styrmodulen. Detta är inget val för gruppen utan är en del av det chassi beställaren har tillhandahållit.

4.3 Styrservo

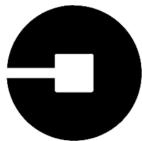
Bilen kommer att ha ett styrservo för framhjulen så att bilen kan justera körriktningen genom att vinkla om hjulen. Detta är inget val för gruppen utan är en del av det chassi beställaren har tillhandahållit.

4.4 Styrreglering

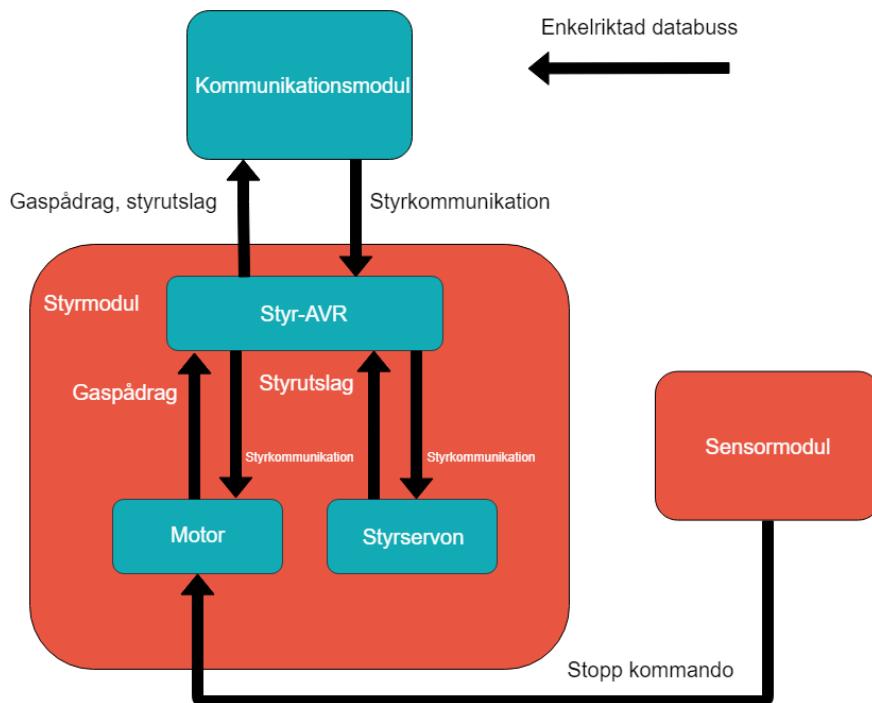
Styrmodulen kommer att ha en regulator med regleringsalgoritmer så bilen kan föras fram på vägen utan att bilen slingrar sig fram eller helt kör av vägen. Regleringen ansvarar även för att bilen, som är framhjulsstyrd, kan svänga korrekt utan att bakhjulen åker utanför banan om en sväng tas för tidigt eller för skarpt.

4.5 Brister

Styrservot för hjulen kan ej vinkla hjulen obegränsat, det vill säga det finns ett maxutslag för styrningen. Detta medför att bilen ej kan ta kurvor som är alltför snäva.



UNTER



Figur 5: Överblick av bilens styrsystem.

5 SENSORMODUL

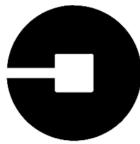
I detta kapitel beskrivs sensormodulen i detalj samt de sensorer som gruppen planerar att använda. För referenser, se figur 6.

5.1 Inledande beskrivning av sensormodulen

Sensormodulen kommer att bestå av en bildbehandlingsdel, en dedikerad AVR-processor och tre externa sensorer: kamera, avståndssensor och vinkelhastighetssensor. Bildbehandlingen kommer att skötas som en process av Raspberry Pi:n och rådatan från avståndssensorn och vinkelhastighetssensorn kommer att behandlas av den dedikerade AVR-processorn. AVR-processorn kommer att skicka data enkelriktat till kommunikationsmodulen i Raspberry Pi:n.

5.2 Kamera

Gruppen planerar att använda en vidvinkelkamera vars uppgift är att läsa av körbanan som specificeras i Appendix A och datan som kommer in (Bilddata) ska tolkas för att kunna urskilja banans linjer. Kameran kopplas direkt till Raspberry Pi:n och kamerans data kommer att behandlas som en process i den. Kommunikationen från kameran kommer att vara enkelriktad (från kameran till Raspberry Pi:n).



UNTER

5.3 Avståndssensor

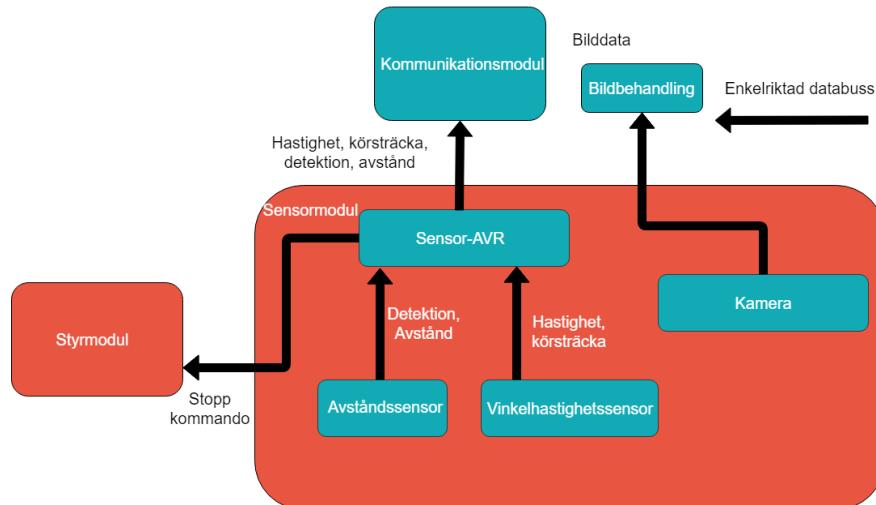
Gruppen planerar att använda en avståndssensor för att detektera fysiska hinder på banan. Daten från avståndssensorn (detektion och avstånd) ska behandlas av den dedikerade AVR-processorn i sensormodulen och ska skickas enkelriktat.

5.4 Vinkelhastighetssensor

Gruppen planerar att använda en vinkelhastighetssensor, som är en hallgivare med magneter, som finns monterad på bilens bakjhul. Daten från vinkelhastighetssensorn (hastighet och körsträcka) ska skickas enkelriktat till och behandlas av den dedikerade AVR-processorn i sensormodulen.

5.5 Brister

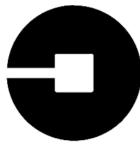
Ett möjligt problem med modulens design kan vara hur avståndssensorn placeras. Om avståndssensorn använder en laser för att mäta avstånd till hinder och sensorn placeras centralt på bilens front kan den eventuellt missa hinder som kommer kort efter en kurva. Gruppen planerar att använda en avståndssensor som mäter avstånd till hinder med hjälp av en bredare laser som kan läsa av ett större område.



Figur 6: Överblick av bilens sensormodul.

6 EXTERN DATOR

Till systemet tillhör en extern persondator som kommunicerar trådlöst med bilden via wifi. Detta delsystem detaljeras ytterligare i figur 7 och 8.



UNTER

6.1 Inledande beskrivning av extern dator

Den externa dator ska kunna ta emot data från bilen under körning och visa denna i ett användargränssnitt (se underkapitlet 6.2). Datorn ska även kunna lagra kördata från bilen, som ska kunna plottas som funktion av tiden i t.ex. MATLAB. Från datorn ska regleralgoritmernas parametrar kunna initieras, det aktuella vägnätet (som ett nodnätverk) ska kunna matas in och det önskade läget på bilen (autonom, semi-autonom eller manuell) ska kunna väljas. Även styrkommandon (i det semi-autonoma läget) och styrning (i det manuella läget) ska kunna skickas.

6.2 Användargränssnitt

Vilket läget som bilen är i påverkar hur användargränssnittet ser ut. Gemensamt för alla lägen är att följande komponenter ska förekomma:

- **Kamera** - I denna komponent ska bilddata från bilens kamera visas i realtid.
- **Konsol** - I konsolen ska information som inte visas i statusfältet kunna visas vid behov. Konsolen ska även tillåta användaren att mata in kommandon, vilket används bland annat för att skicka styrkommandon i det semi-autonoma läget.
- **Status** - I statusfältet visas relevant data, t.ex. bilens batteritid, bilens aktuella körtid och bilens aktuella körräcka.
- **Lägesval** - I denna komponent ska användaren kunna ändra på vilket läge bilen opererar i.

Specifikt i det autonoma och semi-autonoma läget ska även följande komponent finnas:

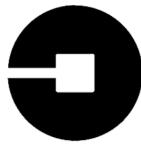
- **Karta** - En karta ska visas som ett nodnätverk. På kartan ska även aktuellt vägsegment visas.
- **Status** - I statusfältet ska lateral position visas utöver den information som visas i statusfältet under manuell körning.

Specifikt i det manuella läget ska även följande komponenter visas:

- **Styrning** - I denna komponent ska den manuella styrningen visas. I det manuella läget styrs bilen med den externa datorns tangentbord, med knapparna W (framåt), A (vänster), S (bakåt) och D (höger). I styrningskomponenten ska riktningen ändra färg så länge den motsvarande knappen är intryckt (i figur 8 är knappen D intryckt vilket motsvarar ett styrutslag åt höger).

6.3 Brister

Den trådlösa kopplingen mellan den externa datorn och bilen kan brytas av olika skäl, t.ex. om avståndet mellan dem blir för stort. Om detta sker ska bilen stanna så fort som möjligt och stå still tills bilen stängs av eller uppkopplingen återupptas.



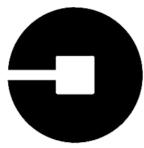
UNTER

Autonombt läge		
Kamera	Karta	Status
		Batteri: 87% Körtid: 12:21 Hastighet: 7 m/s Körsträcka: 182 m Detektion: 84.2 cm Gaspådrag: 32% Styrtslag: -24° Lateral position: 0.1 cm
Konsol	Lägesval	
>	<input checked="" type="checkbox"/> Autonom <input type="checkbox"/> Semi-autonom <input type="checkbox"/> Manuell	

Figur 7: Skiss av användargränssnittet i autonomt läge.

Manuellt läge		
Kamera	Styrning	Status
		Batteri: 87% Körtid: 12:21 Hastighet: 7 m/s Körsträcka: 182 m Detektion: 84.2 cm Gaspådrag: 32% Styrtslag: -24°
Konsol	Lägesval	
>	<input type="checkbox"/> Autonom <input type="checkbox"/> Semi-autonom <input checked="" type="checkbox"/> Manuell	

Figur 8: Skiss av användargränssnittet i manuellt läge.



UNTER

D PROJEKTPLAN



UNTER

Autonom Taxibil

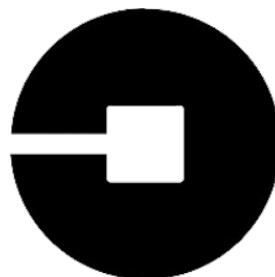
4 juni 2019

Projektplan

Grupp 9

4 juni 2019

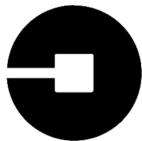
Version 1.0



UNTER

Status

Granskad	JC, OM, EM, NS, DÖ	2019-02-22
Godkänd	Anders Nilsson	2019-02-26

**UNTER****Projektidentitet**Grupp E-post: jonca673@student.liu.se

Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

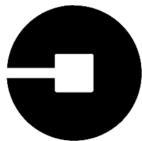
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Designansvarig (DES)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se



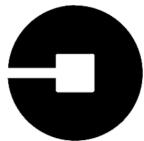
UNTER

INNEHÅLL

1	Beställare	1
2	Översiktlig beskrivning av projektet	1
2.1	Syfte och mål	1
2.2	Leveranser	1
2.3	Begränsningar	2
3	Fasplan	3
3.1	Före projektstart	3
3.2	Under projektet	3
3.3	Efter projektet	3
4	Organisationsplan för hela projektet	3
4.1	Villkor för samarbetet inom projektgruppen	4
4.2	Definition av arbetsinnehåll och ansvar	4
5	Dokumentplan	5
6	Utvecklingsmetodik	6
7	Utbildningsplan	7
7.1	Egen utbildning	7
8	Rapporteringsplan	7
9	Mötesplan	7
10	Resursplan	7
10.1	Personer	7
10.2	Material	7
10.3	Lokaler	8
10.4	Ekonomi	8
11	Milstolpar och beslutspunkter	8
11.1	Milstolpar	8
11.2	Beslutspunkter	8
12	Aktiviteter	9
12.1	Programmering	9
12.2	Styrreglering	10
12.3	Kortastevägalgoritm	10
12.4	Datorseende	11
12.5	Optimering	11
12.6	Tester	12
12.7	GUI-implementering	12
12.8	Övriga aktiviteter	13
12.9	Total fördelning mellan aktiviteter	13
13	Tidplan	13
14	Förändringsplan	13
15	Kvalitetsplan	14
15.1	Granskningar	14
15.2	Testplan	14
16	Riskanalys	14

**UNTER**

17 Prioriteringar	14
18 Projektavslut	15
19 Appendix	15
A Kravspecifikation	15
B Systemskiss	15
C Banspecifikation och tävlingsregler	15
D Tidplan	15

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda ändringar	Utförda av	Granskad
0.1	2019-02-15	Första utkast	Grupp 9	NS
1.0	2019-02-26	Andra utkast	Grupp 9	NS



UNTER

Autonom Taxibil

4 juni 2019

1 BESTÄLLARE

Beställaren av projektet är Anders Nilsson från Institutionen för systemteknik vid Linköpings Universitet.

Tfn: [+46 13282635](tel:+4613282635)

Email: anders.p.nilsson@liu.se

2 ÖVERSIKTlig BESKRIVNING AV PROJEKTET

Detta kapitel beskriver hur projektet utförs i översikt.

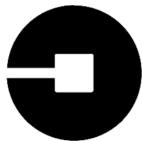
2.1 Syfte och mål

Syftet med projektet är att designa och bygga en prototyp av en autonom bil som kan användas som beslutsunderlag i framtida projekt så som finns beskrivet i appendix A.

Ur ett akademiskt perspektiv så är syftet med projektet att ge medlemmarna erfarenhet inom projektarbete och teknisk problemlösning. Gruppen har därmed som mål att driva projektet på ett sätt som främjar lärande och inkluderar gruppens medlemmar i alla moment.

2.2 Leveranser

I tabell 1 listas datumen för leveranser och delleveranser som ska ske under projektets gång.



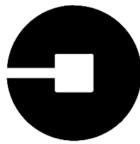
UNTER

Tabell 1: Leveranskrav och delleveranser

Datum	Leverans
2019-02-05	Kravspecifikationen ska vara klar, reviderad och inlämnad till beställaren.
2019-02-15	Första versionen av projektplan, tidplan och systemskiss ska vara inlämnade till beställaren.
2019-02-22	Slutgiltig version av projektplan, tidplan och systemskiss ska vara inlämnade till beställaren.
2019-03-04	Första versionen av gruppens förstudier ska vara skickad till handledare och beställare.
2019-03-08	Första versionen av designspecifikationen ska vara inlämnad till handledaren.
2019-03-15	Designspecifikationen ska vara inlämnad och godkänd av handledaren.
2019-03-27	Version 1.0 av gruppens förstudier ska vara skickade till handledare och beställare.
2019-04-09	Dåvarande design ska vara presenterad för och godkänd av handledaren.
2019-05-07	Demonstration av baskrav, presentation av statusrapporten och diskussion om projektets slutfas, planering och slutkrav ska ske.
2019-05-21	Kappan version 1.0 ska vara inlämnad till beställaren.
2019-05-22	Verifiering av slutkraven.
2019-05-22	Teknisk dokumentation (version 1.0) och användarhandledning (version 1.0) ska vara inlämnade till beställaren.
2019-06-04	Tävlingar och demonstrationer ska ske.
2019-06-10	Efterstudien ska vara inlämnad till beställaren.
2019-06-10	All utrustning och nycklar ska vara återlämnade.
2019-02-25	En tidsrapport som dokumenterar tiden för skrivning av systemskiss och projektplan ska skickas till beställaren.
-	En tidsrapport tillsammans med en uppdaterad tidplan ska skickas till beställare senast kl 16.00 vid följande datum: 2019-03-11, 2019-03-25, 2019-04-01, 2019-04-08, 2019-04-15, 2019-04-29, 2019-05-06, 2019-05-13, 2019-05-20, 2019-05-27, 2019-06-03, 2019-06-10.

2.3 Begränsningar

Bilen ska kunna köras på ett projicerat vägnät i lokalen Visionen. De begränsningar som är satta på bilens köregenskaper beskrivs i appendix C.



UNTER

3 FASPLAN

Denna rubrik beskriver i grova drag de aktiviteterna som ingår i projektets olika faser.

3.1 Före projektstart

Fasen före projektstart innehåller aktiviteter för att få igång projektet. En projektgrupp ska bildas och projektgruppen ska utifrån ett givet projektdirektiv producera en godkänd kravspecifikation. Ansvaret över projektets olika delar ska fördelas inom gruppen och nödvändiga resurser behöver identifieras innan nästa fas kan påbörjas.

3.2 Under projektet

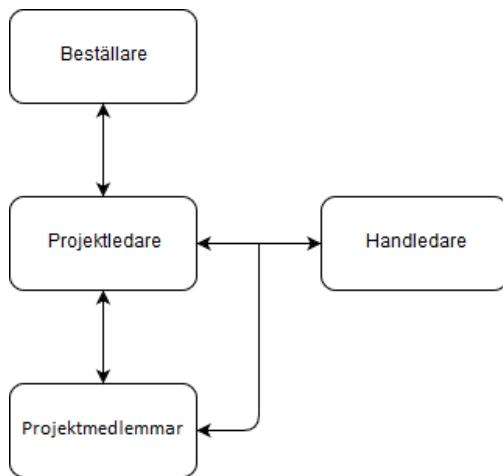
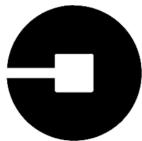
Under projektfasen finns aktiviteter som driver projektet framåt. Projektgruppen ska göra en design för den bil som ska produceras, tester av bilden ska definieras och utföras, hårdvara ska monteras och kod ska skrivas så att bilden uppfyller kraven i appendix A. Under projektets gång ska möten hållas för att se till att projektet drivs i rätt riktning och så att moment kan planeras om när problem uppstår.

3.3 Efter projektet

Efter projektfasen levereras en färdig autonom bil till beställaren som demonstreras i en tävling mot liknande bilar, arrangerad av beställaren. Projektgruppen ska avsluta projektet och lämna tillbaka alla lånade resurser. Tillsist så ska projektgruppen lösas upp.

4 ORGANISATIONSPERSONAL FÖR HELA PROJEKTET

I figur 1 finns en överskådlig bild för hur strukturen ser ut för projektet. Denna rubrik beskriver de olika ansvarsområdena inom projektet mer detaljerat.



Figur 1: En översikt av organisationen i projektet

4.1 Villkor för samarbetet inom projektgruppen

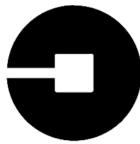
Gruppen diskuterar och tar tillsammans fram gemensamma ordningsregler.

4.2 Definition av arbetsinnehåll och ansvar

I tabell 2 presenteras vilka roller projektets medlemmar har och i tabell 3 presenteras vilka ansvar dessa olika roller har.

Namn	Roller
Jonathan Carlin	Projektledare, Styrmodulansvarig
Olof Mlakar	Dokumentansvarig, Sensormodulansvarig
Emil Mårtensson	Testansvarig, Persondatoransvarig
Nicholas Sepp	Kvalitetssamordnare, Kommunikationsmodulansvarig
Dennis Österdahl	Datorseendeansvarig

Tabell 2: Roller inom projektet



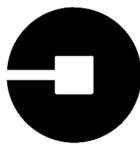
UNTER

Roll	Ansvar
Projektledare	Leda projektgruppen, se till att projektet fortskrider, rapportera till beställare samt sammanställa och skicka in veckovisa tidrapporter.
Dokumentansvarig	Föra mötesprotokoll och organisera projektets olika dokument.
Testansvarig	Ansvarar för testerna av bilen och dess moduler. Se till att testen utförs med förväntat resultat och utforma testrutiner så att tester kan utföras och reproduceras.
Kvalitetssamordnare	Se till att den bestämda kvalitetsnivån i projektet hålls.
Kommunikationsmodulansvarig	Ansvarig för kommunikationsmodulen. Ansvarar för att kommunikationsmodulens funktionalitet är som specificerad i produktdesignen.
Styrmodulansvarig	Ansvarig för styrmodulen. Ansvarar för att styrmodulens funktionalitet är som specificerad i produktdesignen.
Sensormodulansvarig	Ansvarig för sensormodulen. Ansvarar för att sensormodulens funktionalitet är som specificerad i produktdesignen.
Persondatoransvarig	Ansvarig för persondatorn som ska kommunicera med bilen. Ansvarar för att data kan skickas till persondatorn och att denna data lagras och presenteras enligt de krav specificerade i kravspecifikationen.
Datorseendeansvarig	Övergripande ansvar över datorseendet.

Tabell 3: Roller och deras ansvar

5 DOKUMENTPLAN

I tabell 4 listas alla relevanta dokumenten tillhörande projektet. Tabellen beskriver vilket dokument, vem som är ansvarig, vem som godkänner dokumentet, vem som dokumentet distribueras till och när dokumenten ska vara färdigställda. För en mera detaljerad beskrivning av respektive dokument, se appendix A.



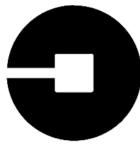
UNTER

Tabell 4: Dokumentplan

Dokument	Ansvarig/godkänns av	Distribueras till	Färdigdatum
Kravspecifikation	Grupp 9 / Anders Nilsson	Anders Nilsson	2019-02-05
Projektplan	Dennis Österdahl, Jonathan Carlin/ Anders Nilsson	Anders Nilsson	2019-02-22
Tidplan	Jonathan Carlin/ Anders Nilsson	Anders Nilsson	2019-02-22
Systemskiss	Nicholas Sepp/ Anders Nilsson	Anders Nilsson	2019-02-22
Designspecifikation	Nicholas Sepp/ Olov Andersson	Olov Andersson	2019-03-15
Förstudie	Nicholas Sepp, Emil Mårtensson, Dennis Österdahl/ Anders Nilsson	Anders Nilsson	2019-03-27
Kappa	Grupp 9/ Anders Nilsson	Anders Nilsson	2019-05-21
Teknisk Dokumentation	Olof Mlakar, Emil Mårtensson/ Anders Nilsson	Anders Nilsson & Opponeringsgrupp	2019-05-22
Användarhandledning	Olof Mlakar, Emil Mårtensson/ Anders Nilsson	Anders Nilsson & Opponeringsgrupp	2019-05-22
Efterstudie	Grupp 9/ Anders Nilsson	Anders Nilsson	2019-06-10
Tidsrapport	Jonathan Carlin/ Anders Nilsson	Anders Nilsson	Veckovis
Mötesprotokol	Olof Mlakar/ Jonathan Carlin	Anders Nilsson vid förfrågan	Veckovis

6 UTVECKLINGSMETODIK

Projektet bedrivs enligt LIPS-modellen och dess projektmetodik. Projektgruppen kommer att använda LIPS som underlag för att organisera och strukturera projektarbetet. Projektet kommer att delas upp i mindre och oberoende aktiviteter med tydligt utsatta krav och mål så att arbetet kan fördelas jämnt inom gruppen och så att projektets olika delar kan utvecklas parallellt med varandra. Projektgruppen kommer ha regelbundna möten och diskussioner för att se till att de olika delarna utförs som planerat och eventuellt omfördela resurser.



UNTER

7 UTBILDNINGSPPLAN

I detta kapitel presenteras den utbildning projektgruppen har fått för att kunna utföra projektet i rimlig tid.

7.1 Egen utbildning

Projektgruppen kommer få utbildning i AVR-processorer och mätteknik. Denna utbildning består av föreläsningar vid Linköpings Universitet samt två laborationer som ger praktisk erfarenhet inom ämnena. Projektgruppen kommer även få en utbildning i etik för att förstå de etiska dilemmarna relaterade till autonoma fordon. Denna utbildning är i form av en föreläsningsserie samt två seminarier.

8 RAPPORTERINGSPLAN

Under projektets gång kommer projektgruppen att rapportera hur projektet fortlöper. Detta kommer att ske i form av tre rapporter, tidrapport, statusrapport och mötesprotokol.

9 MÖTESPLAN

Varje måndag vid lunch hålls möten i en sal bokad av projektledaren. Projektledaren kallar till mötet och dokumentansvarig för mötesprotokoll. Under mötet diskuteras tidsplanen. Gruppen följer upp hurvida tidsplanen följs eller om korrigeringar måste göras. Efter måndagsmötet skickar projektledaren den uppdaterade tidsplanen till beställaren. Projektgruppen har även varje vecka 2 timmar till förfogande att boka in möten med handledaren. Detta kommer att ske kontinuerligt under projektets gång och kommer att, med stor sannolikhet, innebära ett möte i början av arbetsveckan och ett möte i slutet av arbetsveckan.

10 RESURSPLAN

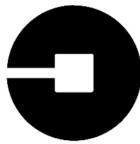
Detta avsnitt beskriver de resurser som kommer användas under projektet.

10.1 Personer

Utförandet av projektet sker av dess projektmedlemmar som består av fem personer. Projektgruppen har en handledare till sitt förfogande som bistår med teknisk expertis. Det kommer även finnas experter inom specifika områden för att ytterligare bistå med kompetens.

10.2 Material

Till projektets förfogande erhålls projektgruppen med ett eget skåp där diverse material och komponenter finns att tillhandahålla för att kunna konstruera den autonoma taxibilen.



UNTER

10.3 Lokaler

Projektet kommer att utföras i B-huset på Linköpings universitet i lokalerna Muxen och Visionen. Konstruktionen av bilen sker i Muxen och tester samt demonstration kommer att ske i Visionen.

10.4 Ekonomi

Ekonomin i projektet anges i tid där varje projektmedlem ska spendera 230 timmar efter BP2.

11 MILSTOLPAR OCH BESLUTSPUNKTER

I denna rubrik presenteras de milstolpar projektgruppen satt upp för projektet och de beslutspunkter som ingår i projektet.

11.1 Milstolpar

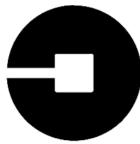
I tabell 5 presenteras de milstolpar som projektgruppen satt upp för projektet.

Tabell 5: Milstolpar i projektet

Nr	Beskrivning	Datum
1	Designspecifikationen godkänd	2019-03-15
2	Datorn ska kummunicera med bilen	2019-03-27
3	Bilen ska köra manuellt	2019-04-03
4	Kamera ska kunna detektera banans linjer (väglinjer, stopplinjer)	2019-04-10
5	Relevant data ska visas och sparas på datorn	2019-04-12
6	Bilen ska kunna köra semi-autonomt	2019-04-17
7	Bilen ska autonomt kunna utföra ett köruppdrag	2019-05-03
8	Bilen ska vara demonstrationsklar	2019-05-07
9	Bilen ska ha ett användarvänligt gränssnitt	2019-05-20
10	Bilen ska vara tävlingsklar	2019-06-04

11.2 Beslutspunkter

I tabell 6 presenteras de beslutspunkter som finns i projektet.



UNTER

Tabell 6: Beslutspunkter i projektet

Nr	Beskrivning	datum
0	Val av projektuppgift	2019-01-25
1	Kravspecifikationen ska vara klar och godkänd av beställaren.	2019-02-05
2	Slutgiltig version av projektplan, systemskiss och tidplan ska vara klara och inlämnade till beställaren.	2019-02-22
3	Designspecifikationen ska vara godkänd av handledaren.	2019-03-15
4	Nuvarande design ska vara presenterad för och godkänd av handledaren	2019-04-09
5a	Demonstration av baskrav, presentation av slutrapporten och diskussion om projektets slutfas planering och slutkrav.	2019-05-07
5b	Verifiering av slutkrav.	2019-05-22
6	Avslutande utav projektet	2019-06-10

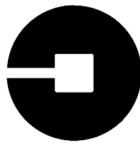
12 AKTIVITETER

I tabellerna nedan presenteras de aktiviteter som projektgruppen satt upp för projektet, indelat i olika huvudområden.

12.1 Programmering

Tabell 7: Aktiviteter med programmering

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
1	Enkel motorstyrning	Styr gasreglage med AVR-processorn från styrmoden	20
2	Enkel servostyrning	Styr hjulservon med AVR-processorn från styrmoden	20
3	Trådlös bilsvängning	Skapa ett sätt att svänga bilen trådlöst från persondatorn	20
4	Trådlöst gasreglage	Skapa ett sätt att styra gasreglage trådlöst från persondatorn	30
5	Trådlös överföring av kördata	Skapa ett sätt för persondatorn att ta emot kördata från bilen	15
6	Enkel dataöverföring från bil till dator	Överföring av data från Raspberry PI till persondatorn	20
7	Körlägen i datorprogram	Implementera funktionalitet i datorprogrammet: byte till nytt körläge	15
8	Styrkommando: Följ vägen rakt fram	Skapa ett nytt styrkommandon i datorprogrammet: "Följ vägen rakt fram"	20



UNTER

12.2 Styrreglering

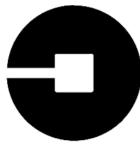
Tabell 8: Aktiviteter som innehåller styrreglering

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
9	Styrreglering i Raspberry Pi m.h.a bilddata	Skapa en funktion som styr bilens servon beroende på bilddata från som kommer från sensormodulen	40
10	Styrkommando: Ta avfart N i nästa rondell	Lägg till ett styrkommando i datorprogrammet som säger att bilen vilken avfart den ska ta i kommande rondell	40
11	Hastighetsreglering vid sväng (Styrmodulen)	Skapa en algoritm som reglerar hastigheten så att bilen har en lämplig fart vid sväng	30
12	Funktionalitet: Stanna bilen vid upptäckt av hinder (styrmodul, sensormodul)	Skapa funktionalitet för styrmodulen att stanna bilen vid upptäckt av hinder	40

12.3 Kortastevägslösningsalgoritmer

Tabell 9: Aktiviteter som innehåller kortastevägslösningsalgoritmer

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
13	Nodnätsnavigation (styrmodulen)	Skapa ett program i styrmodulen som låter bilen navigera efter ett givet nodnät	40
14	Program för Kortaste-väg-beräkningar (styrmodulen)	Skapa en algoritm i styrmodulen som gör kortastevägberäkningar utifrån ett givet nodnät	40



UNTER

12.4 Datorseende

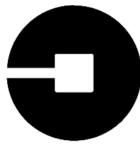
Tabell 10: Aktiviteter som involverar datorseende

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
15	Skapa algoritm i sensormodul som tar behandler bilddata	Läs på om bildbehandling och skapa en algoritm för sensormodulen som presenterar datan från kameran på en form som är lätt att behandla	40
16	Implementera datorseendealgoritm m.h.a. kamera (sensormodul)	Implementera en datorseendealgoritm i sensormodulen som omvandlar bilddata till bilens laterala position på vägen	60
17	Funktionalitet i datorprogram: Visa video från köring	Skapa en algoritm som visar bilddata från en köring i datorprogrammet	20
18	Algoritm: hinderdetektion (sensormodul)	Skapa en algoritm i sensormodulen som omvandlar sensordata till avstånd till (eventuellt) hinder	40

12.5 Optimering

Tabell 11: Aktiviteter som involverar optimering

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
19	Optimering av svängning (inkl tester)	Optimera hastighetsregleringsalgoritmen och svängningsregleringen i styrmodulen för att maximera bilens hastighet i svängar	20
20	Optimering av styrreglering (inkl. tester)	Optimera styrregleringsalgoritmen i styrmodulen för att få bilen att svänga in så snabbt så möjligt och minimera stationära svängningar	20
21	Optimering av kortastvägberäkning (inkl. tester)	Optimera kortastvägberäkningens algoritmen i styrmodulen för att minimera körtid	15
22	Optimering av stoppsträcka vid upptäckt hinder (inkl. tester)	Optimera algoritmen som stannar bilen för att minimera stoppsträckan	20



UNTER

12.6 Tester

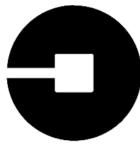
Tabell 12: Aktiviteter som involverar tester

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
23	Test av styrreglering: raksträcka och svängar	Testa och dokumentera bilens förmåga att följa vägen	25
24	Test av styrkommandon under körning	Testa hur bilen reagerar på personatorns styrkommandon när bilen är i manuellt läge.	25
25	Körtest (semiautonomt läge)	Testa och dokumentera bilens prestanda i semiautonomt läge (framförallt stoppträcka vid automatiskt stopp).	20
26	Körtest autonomt läge (endast två noder)	Testa och dokumentera bilens förmåga att köra mellan två slumpmässiga platser	20
27	Körtest autonomt läge (lista av noder)	Testa och dokumentera bilens förmåga att följa slumpmässiga rutter (lista av noder)	25

12.7 GUI-implementering

Tabell 13: Aktiviteter som involverar GUI-implementering

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
28	Skapa GUI för personatorns program	Skapa ett grafiskt gränssnitt som hanterar kommunikation mellan användaren och bilen.	40
29	Användarvänlig manuell styrning	Lägg till användarvänlig funktion i GUI för att svänga med bilen.	10
30	Realtidskarta över bilens position	Generera en karta som visar vilken både i nodnätverket bilen befinner sig på.	10



UNTER

12.8 Övriga aktiviteter

Tabell 14: Aktiviteter som involverar GUI-implementering

Nr	Aktivitet	Beskrivning	Beräknad tid (timmar)
31	Designspecifikation	Skapa projektets designspecifikation.	100
32	Montera bilens moduler	Montera bilens moduler enligt designspecifikationens anvisningar.	20
33	Buffert	Ej planlagd tid för hantering av förseningar	115
34	Möten	Gruppmöten, handledarmöten och beställarmöten.	120

12.9 Total fördelning mellan aktiviteter

Tabell 15: Aktiviteter som involverar GUI-implementering

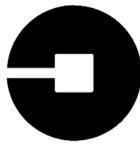
Typ av aktivitet	Beräknad tid (timmar)
Allmän programmering	155
Styrreglering	150
Kortastevägalgoritm	80
Datorseende	160
Optimering	75
Tester	115
GUI-implementering	60
Övrigt	355
Totalt	

13 TIDPLAN

Tidplanen beskriver den tiden projektgruppen har planerat att lägga på de olika aktiviteter som anges i Appendix D.

14 FÖRÄNDRINGSPLAN

Om projektet stöter på problem som leder till förseningar kan krav omförhandlas. Omförhandling måste ske med beställaren och inga krav får ändras eller utgå utan beställarens godkännande. Föreningar i projektdesignen får ske för att lösa eventuella problem. Ändringar i projektdesignen får ej påverka funktionaliteten av andra delar av systemet och ska dokumenteras.



UNTER

15 KVALITETSPLAN

I detta kapitel beskrivs de planer projektgruppen har för att försäkra sig om att kvalitén i projektet håller sig konstant. Projektgruppen anser att väl strukturerade och tydliga dokument kommer underlätta utförandet av projektet och minimera möjliga missförstånd som kan uppstå. Gruppen ämnar att hålla en hög kvalité av skriven kod från början av projektet för att underlätta felsökningar av systemet och för att i slutet kunna leverera en bra produkt.

15.1 Granskningar

Alla dokument projektgruppen skriver ska granskas och godkännas av samtliga projektmedlemmar och slutligen godkännas av gruppens kvalitetssamordnare. Alla kommentarer från granskning av dokument ska behandlas innan dokumentet är godkänt för leverans. All kod som skrivs ska noggrant granskas och testas för att hitta eventuella buggar som kan påverka systemet negativt innan det pushas upp till git.

15.2 Testplan

Test av system och delsystem ska ske kontinuerligt under utvecklandet av bilen. Efter att ett delsystem är klart ska det utförligt testas innan man går vidare och utvecklar nästa del av systemet. Test ska även ske när delsystem som beror av varandra kopplas ihop för att försäkra sig om att inga buggar uppstår och skapar problem senare i projektet. Test ska följa ett testdirektiv framtaget av gruppens testansvarig för att försäkra sig om att test kan reproduceras utan allt för stor varians.

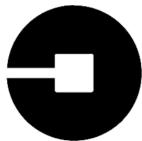
16 RISKANALYS

En möjlig risk inom projektet skulle kunna vara att någon projektmedlem inte utför sin tänkta arbetstid. Detta medför då att projektet blir försenat och i värsta fall inte är möjligt att genomföra. Genom att projektgruppen har en öppen dialog med varandra och högt i tak gentemot varandras åsikter leder detta till en öppen och ärlig grupp där fokus hamnar på att man kan vara sig själv och således stärka känslan av tillhörighet vilket medför en vilja att fortsätta arbetet med projektet tillsammans med gruppen.

En annan risk som skulle kunna påverka projektets fortlöpande är risken att komponenter i bilen går sönder. Detta skulle leda till att projektet tar längre tid att utföra och att det inte hinner klart inom den bestämda tiden. Genom en väl dokumenterad designspecifikation, väl kommenterad kod, noggrann konstruktion och att alla inom projektgruppen har en god helhetssyn över bilen och dess komponenter leder detta till en konstruktion av bilen där risken för att något går sönder minimeras. Att även under projektets gång göra kontinuerliga tester säkrar kvalitén.

17 PRIORITERINGAR

För att strukturera upp genomförandet av projektet har en prioriteringsnivå implementerats. Denna nivå har en rankad skala från 1-3 där 1 benämner den högsta prioriteten och 3 den lägsta prioriteten. Alla krav som sätts på bilen har erhållits en prioriteringsnivå, för vidare information se appendix A. Skulle det inträffa förseningar inom projektet är



UNTER

det alltid de krav med lägst prioritet som prioriteras först. De system med högst prioritet är de moduler som gör bilen körbar, sensormodul, styrmodul och kommunikationsmodul. Skulle förseningar dyka upp på någon aktivitet som anses kritisk för projektets fortlöpande kommer resurser att omföredelas för att lösa aktiviteten.

18 PROJEKTAVSLUT

Projektet avslutas med en tävling arrangerad av beställaren där bilen tävlar mot liknande bilar enligt de specificerade tävlingsreglerna i appendix C. Efter tävlingen och om beställaren är nöjd ska projektgruppen skriva en efterstudie samt lämna in den kompletta kandidatrapporten och all källkod för projektet. Till slut ska projektgruppen lämna tillbaka alla lånade resurser och tillsist ska projektgruppen upplösas.

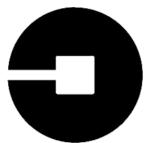
19 APPENDIX

A KRAVSPECIFIKATION

B SYSTEMSKISS

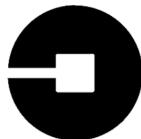
C BANSPECIFIKATION OCH TÄVLINGSREGLER

D TIDPLAN



UNTER

E DESIGNSPECIFIKATION



UNTER

Autonom taxibil

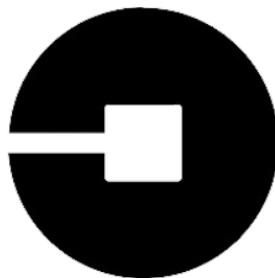
14 mars 2019

Designspecifikation

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl

14 mars 2019

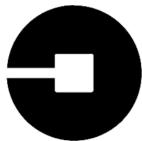
Version 0.2



UNTER

Status

Granskad	Nicholas Sepp	2019-03-14
Godkänd		2019-xx-xx

**UNTER****Projektidentitet**Grupp E-post: jonca673@student.liu.se

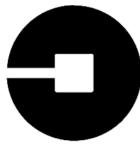
Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

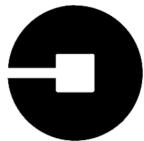
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Datorseendeansvarig (DAT)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se



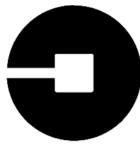
UNTER

INNEHÅLL

1	Inledning	1
1.1	Syfte	1
1.2	Definitioner	1
2	Översikt av systemet	1
3	Kommunikationsmodul	4
3.1	Beskrivning av systemet	4
3.2	Kopplingsschema	5
3.3	Trådlös kommunikation	6
3.4	Busskommunikation mellan processorer	7
3.5	Parallella processer	9
3.6	Kortaste väg-algoritm	9
4	Styrmodul	11
4.1	Beskrivning av systemet	11
4.2	Kopplingsschema	12
4.3	Komponenter	13
4.3.1	Mikroprocessor - ATmega1284P	13
4.3.2	Styrservo	13
4.3.3	DC-motor	14
4.4	Pulsbreddsomvandlare	14
4.5	JSP diagram för styalgoritmen	15
5	Sensormodul	15
5.1	Beskrivning av modulen	16
5.2	Kopplingsschema	16
5.3	Komponenter	17
5.3.1	Mikroprocessor - ATmega1284P	17
5.3.2	Ultraljudssensor - SRF04	18
5.3.3	Halleffektsensor - A1120	19
5.3.4	Kamera - RPI Camera	20
5.4	JSP-diagram för avståndssensor	21
5.5	JSP för HallEffektSwitch	21
5.6	JSP-diagram för datorseendet	21
6	Persondator	22
6.1	Lagring av data	22
6.2	Grafiskt gränssnitt	23
7	Appendix	25

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda ändringar	Utförda av	Granskad
0.1	2019-03-08	Första utkast	Grupp 9	NS
0.2	2019-03-14	Andra utkast	Grupp 9	JC



UNTER

1 INLEDNING

Inför *under-* fasen i projektkursen TSEA56 ska en designspecifikation, som detaljerat beskriver den tänkta konstruktionen och programmeringen, skrivas. I kapitel 2-6 går gruppen igenom hur varje huvudmodul, samt persondator, ska konstrueras och programmeras.

1.1 Syfte

Syftet med dokumentet är att gruppen ska ha en utgångspunkt på hur projektet ska genomföras praktiskt. Dokumentet är inte bindande och förändringar under projektets gång kan förekomma, dock så ska dessa helst göras i samtal med handledaren. Dokumentet ska granskas och godkännas av gruppens handledare (Olov Andersson) och när handledaren ger klartecken så kan gruppen påbörja byggandet av den autonoma taxibilen.

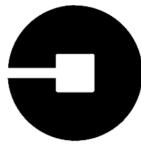
1.2 Definitioner

Nedan följer förklaringar till förkortningar som används i dokumentet.

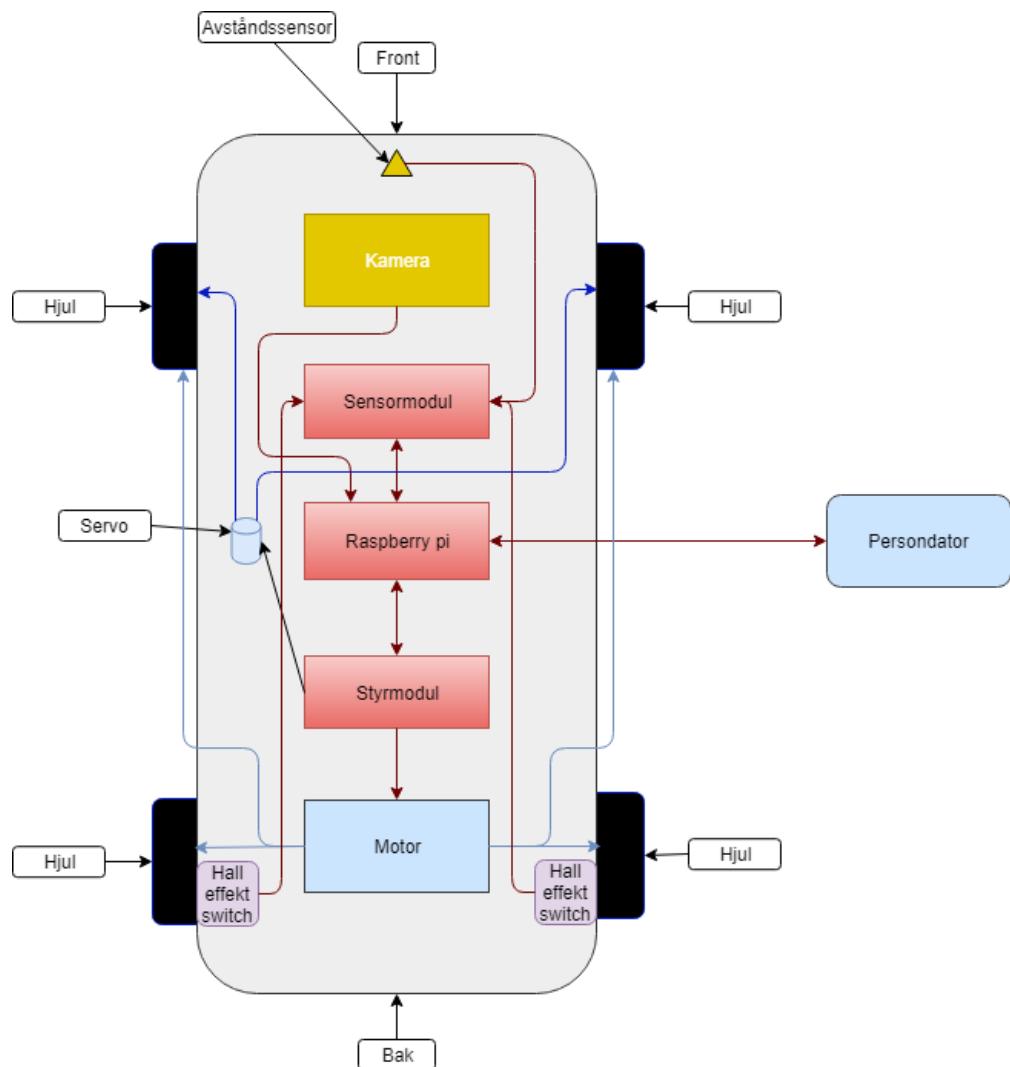
- RPI: Raspberry Pi 3 Model B[1], enkortsdator.
- AVR: Familj av mikrokontroller som används av två av projektets moduler.
- JTAG: Interface från datorn till AVR-processorn.
- GUI: Grafiskt användargränssnitt (eng: Graphical User Interface).

2 ÖVERSIKT AV SYSTEMET

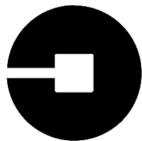
Systemet i dess helhet beskrivs i figur 1 och i figur 2 erhålls en överblick över alla delsystem som ingår. Då två delsystem kommer använda sig av mikroprocessorn ATmega1284P kan man i figur 3 se alla dess pinnar.



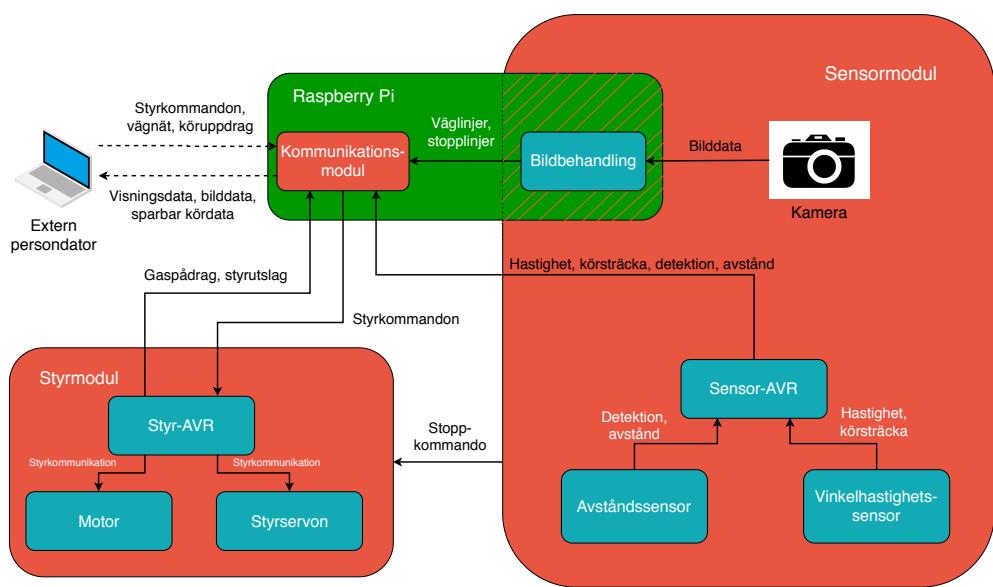
UNTER



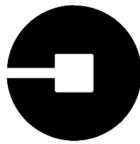
Figur 1: Överblick av bilen och dess hårdvara sett ovanifrån.



UNTER



Figur 2: Överblick av bilens delsystem.



UNTER

(PCINT8/XCK0/T0)	PB0	1	40	PA0 (ADC0/PCINT0)
(PCINT9/CLK0/T1)	PB1	2	39	PA1 (ADC1/PCINT1)
(PCINT10/INT2/AIN0)	PB2	3	38	PA2 (ADC2/PCINT2)
(PCINT11/OC0A/AIN1)	PB3	4	37	PA3 (ADC3/PCINT3)
(PCINT12/OC0B/SS)	PB4	5	36	PA4 (ADC4/PCINT4)
(PCINT13/ICP3/MOSI)	PB5	6	35	PA5 (ADC5/PCINT5)
(PCINT14/OC3A/MISO)	PB6	7	34	PA6 (ADC6/PCINT6)
(PCINT15/OC3B/SCK)	PB7	8	33	PA7 (ADC7/PCINT7)
<hr/>		9	32	AREF
VCC		10	31	GND
GND		11	30	AVCC
XTAL2		12	29	PC7 (TOSC2/PCINT23)
XTAL1		13	28	PC6 (TOSC1/PCINT22)
(PCINT24/RXD0/T3)	PD0	14	27	PC5 (TDI/PCINT21)
(PCINT25/TXD0)	PD1	15	26	PC4 (TDO/PCINT20)
(PCINT26/RXD1/INT0)	PD2	16	25	PC3 (TMS/PCINT19)
(PCINT27/TXD1/INT1)	PD3	17	24	PC2 (TCK/PCINT18)
(PCINT28/XCK1/OC1B)	PD4	18	23	PC1 (SDA/PCINT17)
(PCINT29/OC1A)	PD5	19	22	PC0 (SCL/PCINT16)
(PCINT30/OC2B/ICP)	PD6	20	21	PD7 (OC2A/PCINT31)

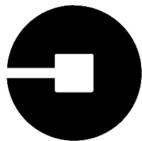
Figur 3: Kopplingsschema ATmega1284P.

3 KOMMUNIKATIONSMODUL

Denna sektion redogör för kommunikationsmodulens uppbyggnad och funktionalitet. Figur 4 visar en övergripande bild av modulen.

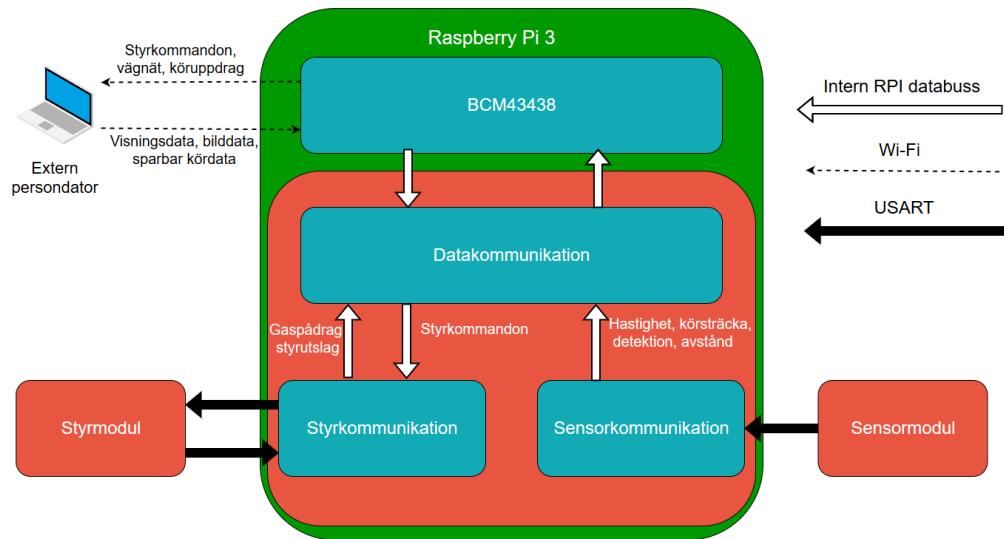
3.1 Beskrivning av systemet

Kommunikationsmodulen ansvarar för kommunikation mellan styrmodulen, sensormodulen och den externa persondatorn. Den externa persondatorn ska visa data som tillhandahålls av både styr- och sensormodulen och så ska persondatorn kunna skicka data till övriga moduler. T. ex. ska körkommandon från persondatorn kunna skickas via styrmodulen till bilen i manuellt läge som i sin tur svarar med relevant köldata via sensormodulen såsom körtid och körsträcka. Data som ska från en modul till persondatorn kommer först att överföras till kommunikationsmodulen via en buss (se underrubrik 3.4) för att sedan trådlöst överföras till datorn. Data från datorn till en modul sker självklart på samma sätt fast åt motsatt håll. Kommunikationsmodulen är i sin helhet implementerad som en process på RPI:n (se figur 2). Kommunikationsmodulen utnyttjar mikrodatorns nätverksmodul (BCM43438)[1] för att trådlöst kommunicera med



UNTER

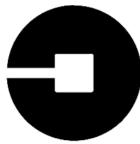
persondatorn via Wi-Fi och två av mikrodatorns USB-portar för att kommunicera via USART med de andra två modulernas AVR-processorer.



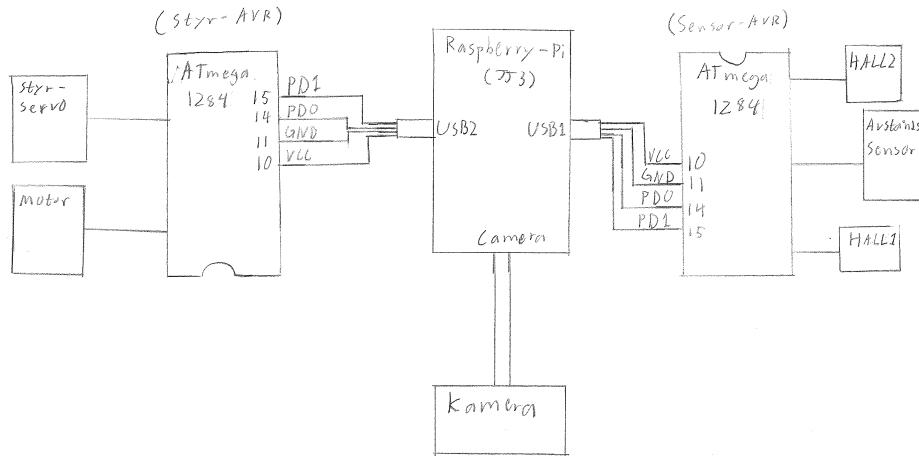
Figur 4: Systemskiss av kommunikationsmodulen.

3.2 Kopplingsschema

Figur 5 visar hur alla moduler och kameran är tänkta att kopplas ihop med RPI:n. Kameran kopplas till en dedikerad kameraport på mikrodatorn. Till två av mikrodatorns USB-portar kopplas FTDI TTL USB till UART kablar [2] för att möjliggöra kommunikation mellan mikrodatorn och de andra två modulernas AVR-processorer via USART. Från respektive kabel kopplas andra änden till pinnarna PDO (RXD) och PD1 (TXD), som ansvarar för att mottaga och sända data respektive, samt matningsspänning på 5 V och jord. Sensormodulen och styrmodulen detaljeras i sektionerna 5 och 4 respektive.



UNTER



Figur 5: Kopplingsschema för RPI:n, kameran och de andra två modulernas AVR-processorer.

3.3 Trådlös kommunikation

För att kunna trådlöst kommunicera med RPI:n ska den användas som en anslutningspunkt i ett fristående nätverk. Hur detta ska göras detaljeras steg för steg i dokumentationen för RPI [3]. De viktigaste principerna redogörs för nedan. Följande görs under antagandet att RPI:n använder sig av operativsystemet Raspbian (version från 2017 eller senare). RPI:n ska agera som server i ett fristående nätverk, så en statisk IP adress bör tilldelas dess trådlösa port. Det antas att standard 192.168.xx IP adresser används och att den trådlösa porten som används är wlan0. För att konfigurera den statiska IP adressen ska dhcpcd konfigurationsfilen ändras så att följande står i slutet av filen

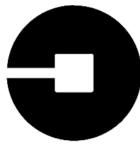
```
interface wlan0
    static ip_address=192.168.4.1/24
    nohook wpa_supplicant
```

DHCP-tjänsten tillhandahålls av dnsmasq. Standardkonfigurationsfilen innehåller mycket information som inte kommer vara nödvändigt, så en ny dnsmasq konfigurationsfil bör skapas. I denna ska följande stå

```
interface=wlan0      # Use the require wireless interface - usually wlan0
    dhcp-range=192.168.4.2,192.168.4.20,255.255.255.0,24h
```

Det vill säga att wlan0 tilldelas IP adresserna mellan 192.168.4.2 och 192.168.4.20, var med en lease-tid på 24 timmar. Anslutningspunktens värdmjukvara (access point host software, hostapd) ska konfigureras i hostapd konfigurationsfilen. Följande konfiguration använder kanal 7, UNTER som nätverksnamn och 0000 som lösenord. Variabeln hw_mode bestämmer vilket IEEE 802.11 protokoll som används, denna konfiguration använder 802.11g.

```
interface=wlan0
driver=n180211
ssid=UNTER
hw_mode=g
channel=7
wmm_enabled=0
```



UNTER

```
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=0000
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Efter ytterligare några konfigurationer ska RPI:n vara en anslutningspunkt. I början av utvecklingen kommer åtkomst till RPI:n från den externa persondatorn att främst ske via SSH [4], tills att det grafiska gränssnittet har blivit utvecklat, se underrubrik [6.2](#).

3.4 Busskommunikation mellan processorer

Kommunikationen mellan RPI:n och övriga modulers processorer (två AVR-processorer) sker via USART. USART är ett seriellt gränssnitt som kan programmeras till att kommunicera asynkront och till skillnad från UART även synkront. På AVR ATMega1284P-processorn finns två USARTs, USART0 och USART1, endast USART0 ska användas på båda processorer. På processorna är pinnarna PD0 och PD1:s alternativa funktioner RXD0 och TXD0, vilka används för att mottaga och sända data via USART0 respektive. Dessa alternativa funktioner för pinnarna aktiveras genom att sätta korresponderande bitar RXEN0 och TXEN0 i kontrollregistret UCSR0B till 1.

```
#include <avr /io.h>
int main (void)
{
    UCSR0B = (1 << RXEN0) | (1 << TXEN0); // Turn on the transmission and reception circuitry
}
```

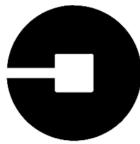
USART stödjer fyra olika lägen: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous. USART0 ska endast användas till asynkron kommunikation, vilket betyder att en extern synkron klocka inte behöver användas. I detta fall beter sig bussen som en UART. Normalt asynkront läge väljs genom att sätta UMSEL00-biten i kontrollregistret UCSR0C till 0 och U2X0-biten i kontrollregistret UCSR0A till 0, vilket är bitarnas standardinställningar. UMSEL00 styr huruvida USART0 är asynkron eller synkron och U2X0 styr huruvida USART0 är i läget Normal asynchronous eller Double Speed asynchronous.

Kommunikationen mellan enheter upprättas genom att man förbestämmer en vald baud rate, baud rate:n beskriver kommunikationshastigheten i bitar per sekund på bussen. En förbestämd klockfrekvens väljs även på processorns systemklocka detta då den används för att sampla TXD0 och RXD0 i precisa intervaller för att kunna uppehålla kommunikationen. När dessa väl är valda behöver man ställa in baud rate registret UBR for att erhålla vald baud rate. Ekvation [1](#) beskriver hur detta värde tas fram.

$$UBRR0 = \frac{F_{CPU}}{16BAUD} - 1 \quad (1)$$

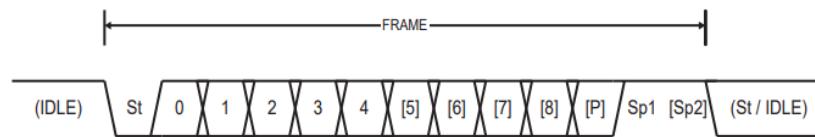
Detta värde finner man i processorns datablad men genom att tillämpa ekvation [1](#) kan man implementera den som ett makro och behöver således inte hårdkoda in siffror i källkoden. Vid vald klockfrekvens på 1.8432 MHz och en baud rate på 9600 bps erhålls värdet 11. Detta värde sätter man sedan register UBR till och upprättar då en kommunikationshastighet på bussen med en buad rate på 9600 bps. Koden som genererar UBR är som följande

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / 16) + (USART_BAUDRATE / 2)) / (USART_BAUDRATE )) - 1
```



Makrot matchar inte ekvationen 1 helt och hållet, detta eftersom denna variant avrundar UBRR0 på ett sådant sätt som säkerställer att baud rate:n blir så nära det önskade värdet som möjligt.

Ett seriellt frame format ska definieras för att överföra information på bussen. ATmega1284P-processorn stödjer 30 olika kombinationer av startbit, databitar, paritetsbitar och stoppbitar, se figur 6. Frame formatet som ska användas består av en startbit, 8 databitar och en stoppbit. Antalet databitar i en frame bestäms av UCSZ01:0-bitarna i kontrollregistret UCSR0C och UCSZ02-biten i UCSR0B. Antalet stoppbitar bestäms av USBS0-biten och paritetsläget bestäms av bitarna UPM01:0 i kontrollregistret UCSR0C. En komplett initialisering av USART0 visas i koden 1.



Figur 6: Möjliga kombinationer av frame-format för ATMega1284P. Bitar i hakparanteser är valfria.

```
#include <avr /io.h>

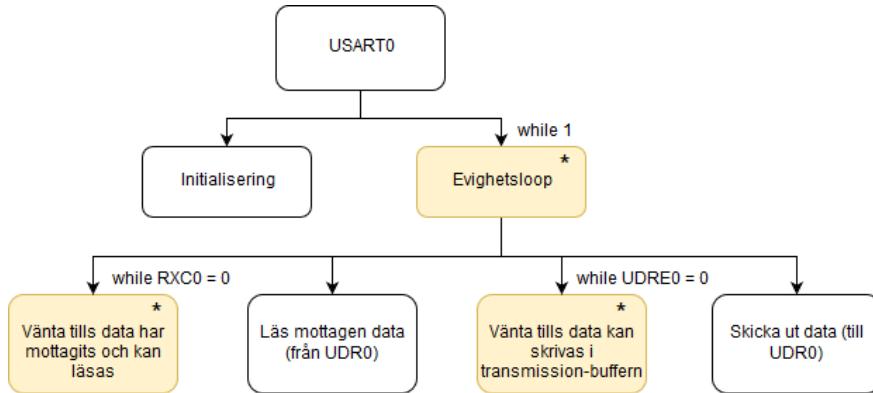
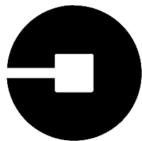
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / 16) + (USART_BAUDRATE / 2)) / (USART_BAUDRATE )) - 1

int main (void)
{
    UCSRB = (1 << RXEN0) | (1 << TXEN0); // Turn on the transmission and reception circuitry
    UCSRC = (1 << UCSZ00) | (1 << UCSZ01); // Set frame format: 8 data bits, 1 stop bit

    // Set the baud rate registers
    UBRRH = (BAUD_PRESCALE >> 8); // Load upper 8- bits of the baud rate value into UBRR0H
    UBRRL = BAUD_PRESCALE; // Load lower 8- bits of the baud rate value into UBRRL
}
```

Listing 1: Initialisering av USART0 på en ATmega1284P.

I figur 7 visas ett kort exempel på hur data kan mottagas och skickas via USART som ett JSP-diagram. Rutan Initialisering innebär en initialisering av USART0 som i koden 1.



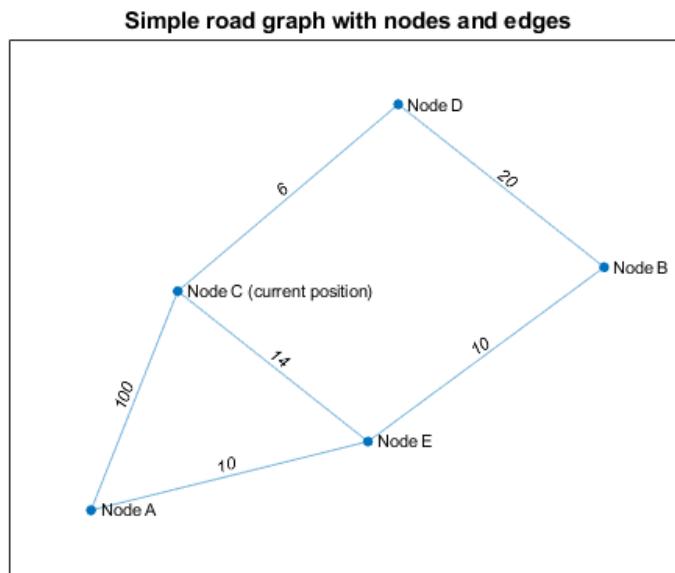
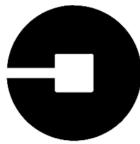
Figur 7: Ett exempel på hur USART-kommunikation kan gå till.

3.5 Parallella processer

Då en autonom taxibil kan betraktas som ett system där informationen kräver ett snabbt uppdateringsflöde är det viktigt att inte skapa ett program vars operationer sker i en löpande takt. Detta leder nämligen till att många operationer får vänta på att andra operationer ska bli färdiga. Genom att skapa parallella processer i kommunikationsmodulen uppnår man parallelism och flera saker kan utföras samtidigt. När man nyttjar parallella processer är det viktigt att hålla koll på att rätt information hanteras av rätt process. Om flera processer samtidigt vill läsa från eller skriva till en gemensam resurs kan det uppstå problem i form av exempelvis överkrsivningar av den gemensamma resursen. Detta kan leda till att kommunikationsmodulen levererar fel data till fel modul. Genom att implementera en ömsesidig uteslutning, även kallad mutex, förhindrar man att flera processer kan få tillgång till gemensamma resurser samtidigt.

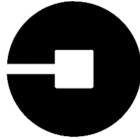
3.6 Kortaste väg-algoritm

Bilen behöver i sitt köruppdrag tolka en lista med destinationer för att ta fram en väg att köra. Ett vägnät med vägar och platser beskrivs med en oriktad, viktad graf (se figur 8 nedan). Vägnätet som bilen designas för innehåller endast rondeller, inga korsningar. Varje nod förbinds av bågar som har en siffra som beskriver kostnaden för att använda bågen, i detta fall den fysiska sträckan mellan två noder. En nod beskriver både hållplatser och rondeller, och en nod med över 2 bågar är en rondell med lika många avfarter.

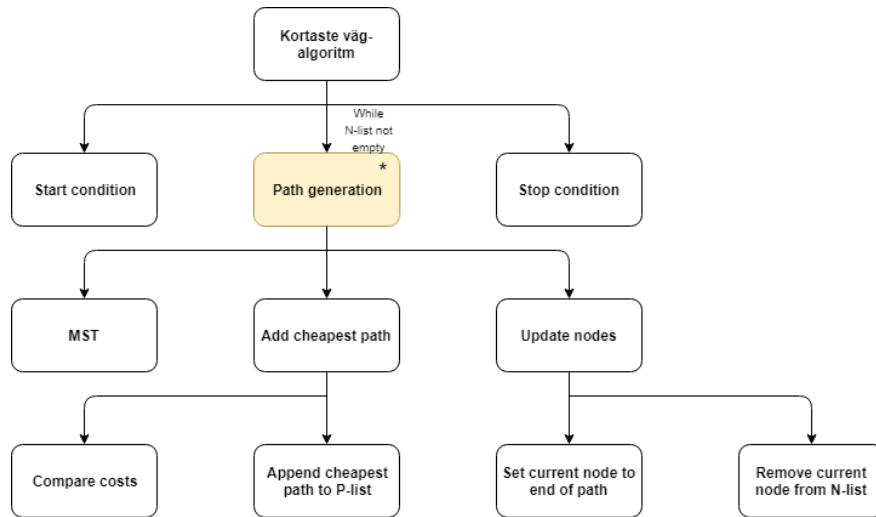


Figur 8: Oriktad graf som beskriver vägnätets struktur

Det finns bättre och sämre vägar för bilen att välja för att minimera körsträckan. För att ta fram den kortaste vägen till alla noder kan man skapa ett minsta uppspänande träd (MST), detta träd använder en specifik nod som startpunkt. Man kan iterativt beräkna en optimal rutt som bilen ska följa för att besöka ett antal utvalda noder på kortast möjliga körsträcka. JSP-diagrammet nedan (figur 9) beskriver översiktligt hur man tar fram denna rutt. Denna algoritm använder en nodkarta och en lista med noder som ska besökas (N-list), för att ta fram en färdig lista med bågar (P-list) som bilen ska följa för att besöka noderna på kortast möjliga sätt.



UNTER



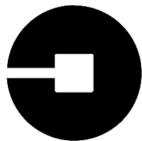
Figur 9: JSP-diagram för kortaste väg-algoritmen

4 STYRMODUL

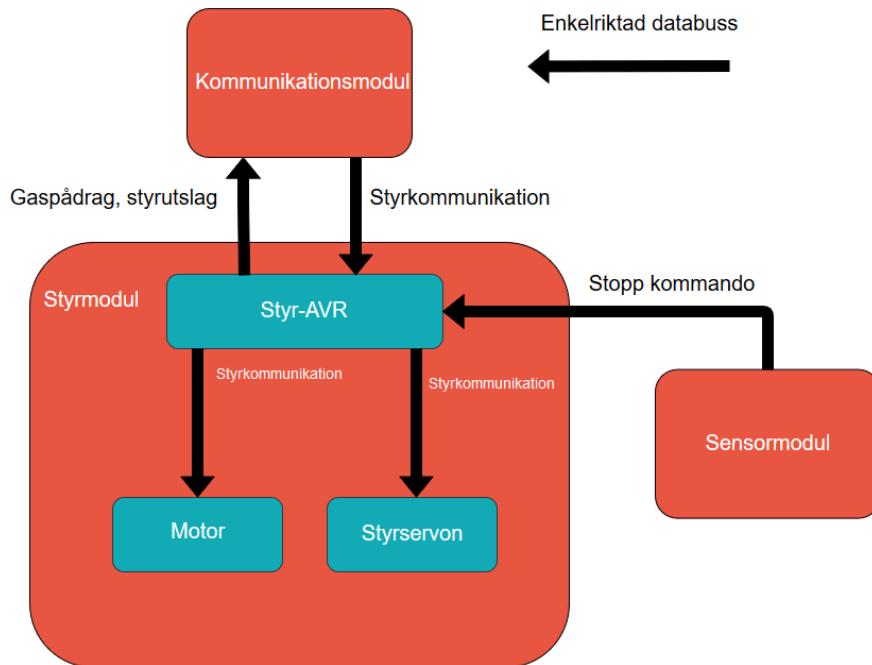
Denna sektion redgör för styrmodulens uppbyggnad och funktionalitet.

4.1 Beskrivning av systemet

Styrmodulens uppgift är att styra bilens hastighet och styrutslag. Detta sköts genom att två pulsbreddsmudulerade spänningar med frekvensen 50 Hz ($f_{servo} = 50\text{Hz}$) skickas till styrservot respektive motorn. I figur 10 visas en överblick av styrmodulen. Modulen utnyttjar processorn ATmega1284P för regleringen av gasreglage och styrutslag. Modulen tar emot styrkommandon från kommunikationsmodulen via en USART-buss. Styrmodulen ansvarar även för att skicka information om den använda pulsbredden för gaspådrag och styrutslag till kommunikationsmodulen.



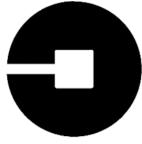
UNTER



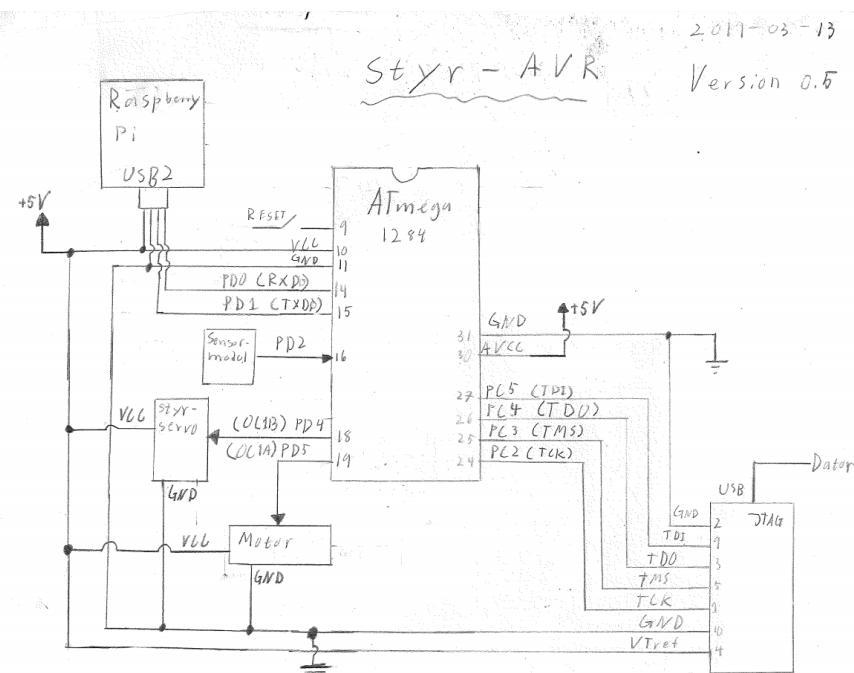
Figur 10: Överblick av bilens styrsystem.

4.2 Kopplingsschema

Figur 11 beskriver styr-AVR:ens tänkta uppkoppling. I figuren är JTAG:en (som används för programmering av AT-megan) inkopplad, i den färdigkopplade versionen av bilen så kommer JTAG:en ej vara inkopplad. Kopplingsschemat baseras på en ATmega1284P processor med pinnummer som motsvarar de i figur 3. Busskomunikationen kommer att ske via en USART från RPI:n (beskrivs i avsnitt 3.4), kommer att kopplas till PD0 (RXD0) och PD1 (TXD0), enligt specifikationer, samt jord och matningsspänning. PD4 och PD5 används för att skicka in styrdata till servot respektive motorn i form av pulsbredder mellan 1.0 ms och 2.0 ms med periodtiden 20 ms. PD2 (INT0) är ett externt interrupt som grupperna planerar att koppla direkt till sensor-AVR:en, om sensormodulen upptäcker ett fysiskt hinder med ultraljudsensorn (beskrivs i avsnitt 5.3.2) så ska avbrott som aktiverar bilens inbromsning aktiveras. Utöver dessa pinnar så ska VCC, AVCC (matningsspänning +5V), GND (jord) och RESET kopplas in.



UNTER



Figur 11: Kopplingsschema Styr-AVR.

4.3 Komponenter

Denna sektion går igenom hårdvaran som styrmodulen består av, samt pulsbreddsomvandlaren i ATmega:n. Styrmodulen består av följande hårdvarukomponenter (JTAG exkluderad).

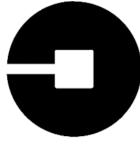
- 1 ATmega1284P
- 1 Styrservo
- 1 DC-motor

4.3.1 Mikroprocessor - ATmega1284P

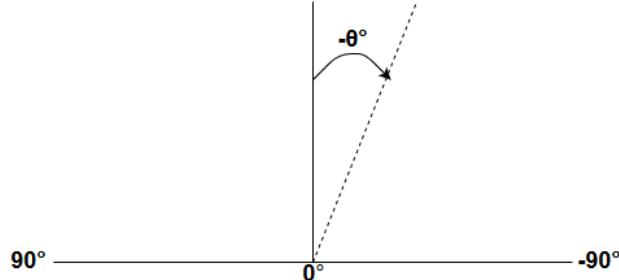
En ATmega1284P används för alla styrrelaterade beräkningar. Den har valts istället för en ATmega16 för att underlätta implementeringen i projektet då en ATmega1284P krävs för sensormodulen (se avsnitt 5.3.1). En ATmega16 fungerar för styrmodulens uppgifter men då projektets budget räknas i tid och inte i pengar så har ATmega1284P ändå valts då implementeringen blir snarlik den för sensormodulen och förbättrar således tidsbudgeten. ATmega1284P har två 16-bitars timers för pulsbreddsomvandling (se avsnitt 4.4) vilket behövs för styrservon och motorn. Den sköter alltså båda regleralgoritmerna och pulsbreddsmoduleringen till motorn respektive servot.

4.3.2 Styrservo

Bilens styrservo bestämmer framhjulens styrvinkel. Servot kräver som insignal en pulsbreddsmodulerad spänning med frekvensen 50 Hz. Den kortaste pulsbredden som servot kan ta emot är 1.0 ms, motsvarar en utfallsvinkel på 90°, och



den längsta är 2.0 ms, motsvarar -90° . Mittpunkten för styrutslaget är 1.5 ms och motsvarar en utfallsvinkel på 0° . Se figur 12 för referens.



Figur 12: Beskrivning av styrutslag.

4.3.3 DC-motor

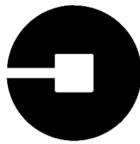
Bilen är fyrfjulsdriven och drivs av en DC-motor som fungerar som styrservon i avsnitt 4.3.2. Den har en periodtid på 20 ms (50 Hz) och regleras med en pulsbreddsomvandlare. Den kortaste pulsbredden som servot kan ta emot är 1.0 ms och den längsta är 2.0 ms. En pulsbredd på 1.5 ms innebär att bilen står still, en pulsbredd som är mindre än 1.5 ($x < 1.5$) innebär att bilen backar och en pulsbredd som är större än 1.5 ms ($x > 1.5$) innebär att bilen kör framåt.

4.4 Pulsbreddsomvandlare

Gruppen tänker använda en klockfrekvens på $f_{clock} = 1.8432MHz$ på mikroprocessornerna. För att implementera en periodtid på 20 ms så måste antalet klockcykler under den tiden (*Ticks*) tas fram, se ekvation 2. Då 16-bitars timern används med ett 16-bitars register ($2^{16} - 1 = 65535$) behöver utsignalen inte prescalas, se ekvation 3, vilket betyder att vi minskar risken att få fel i pulsbredden som skickas till motorn och styrservot. För pulsbreddsmoduleringen kommer metoden *Fast-PWM* att användas.

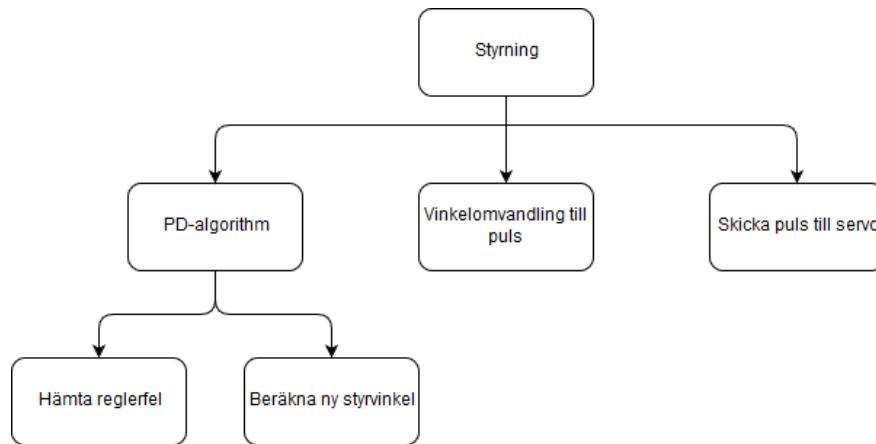
$$Ticks = \frac{f_{clock}}{f_{servo}} - 1 = \frac{1.8432 * 10^6}{50} - 1 = 36863 \quad (2)$$

$$36863 < 65535 \quad (3)$$



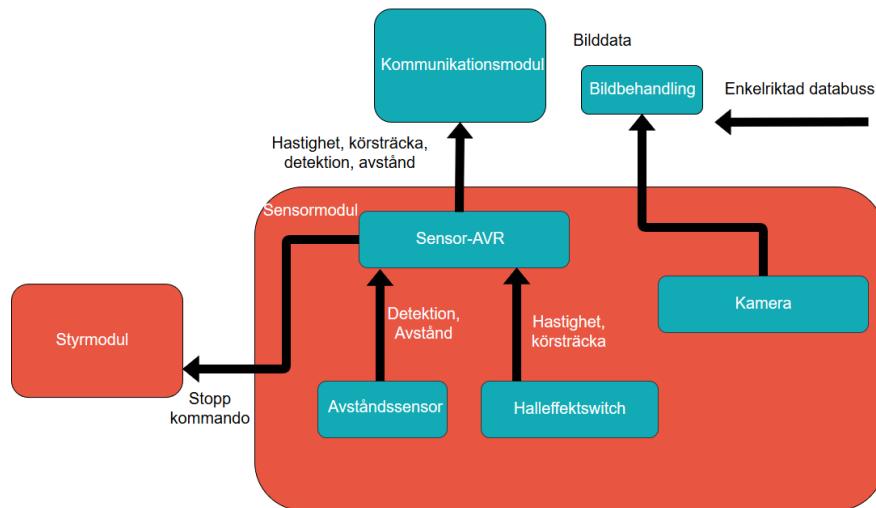
UNTER

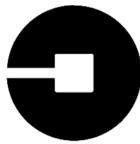
4.5 JSP diagram för styralgoritmen

**Figur 13:** JSP-diagram för modulens styralgoritmen

5 SENZORMODUL

Denna sektion behandlar sensormodulens uppbyggnad. Figur 14 visar en övergripande systemskiss på sensormodulens funktionalitet.

**Figur 14:** Överblick av bilens sensormodul.



UNTER

5.1 Beskrivning av modulen

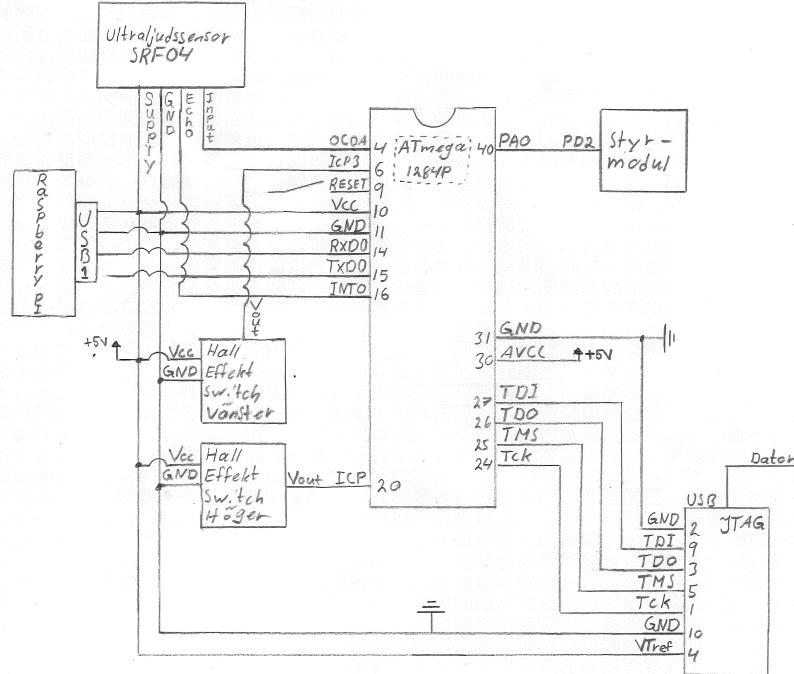
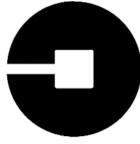
Det är sensormodulens uppgift att samla in data för att åstadkomma ett lyckat köruppdrag. Genom att installera en RPI-kamera kommer bilen kunna lyckas detektera väglinjer och således ha möjligheten att köra på vägen. Tillsammans med kameran och programbiblioteket OpenCV kommer datorseende algoritmer att implementeras för att kunna upptäcka väglinjerna. På vägnätet kommer även hinder att placeras ut som bilen inte får krocka med. För att inte krocka med hindrena på vägen kommer en ultraljudssensor att installeras på bilen. Ultraljudssensorn mäter avståndet till hindret och ger då bilen möjlighet att kunna beräkna avståndet till bilen. Det är även viktigt att veta i vilken hastighet man kör i och hur långt man har åkt. En halleffektswitch kommer att installeras på vardera bakhjul. Med hjälp av halleffektswitchen kan man räkna ut bilens hastighet samt hur långt den har kört.

Både ultraljudssensorn och halleffektswitchen kommer att kopplas in i en mikroprocessor, ATmega1284P, för behandling av den insamlade sensordatan och sedan skickas med en USART-buss till kommunikationsmodulen som ligger direkt på RPI-enheten. RPI kameran kopplas direkt in på RPI:ns dedikerade kameraingång. Kamerans bildbehandling sker också på RPI:n.

5.2 Kopplingsschema

Figur 15 beskriver sensor-AVR:ens tänkta uppkoppling. I figuren är JTAG:en (som används för programmering av ATmega:n) inkopplad, i den färdigkopplade versionen av bilen så kommer JTAG:en ej vara inkopplad. Kopplingsschemat baseras på en ATmega1284P processor med pinnummer som motsvarar de i figur 3. Busskomunikationen kommer att ske via en USART från RPI:n (beskrivs i avsnitt 3.4), kommer att kopplas till PD0 (RXD0) och PD1 (TXD0), enligt specifikationer, samt jord och matningsspänning.

De båda halleffektswitcharna kommer att kopplas till Atmegans båda input capture pinnar. Ultraljudssensorn kommer få en input från OC0A som kommer att fungera som en pulsbreddsmodulering. Ultraljudssensorn utsignal till ATMegan kommer att skickas till externt avbrott, pin INT0. Från pin PA0 kommer en utsignal till styrmodulen att kopplas. Denna kan ses som en nädbroms och kommer att skicka ett avbrott till styrmodulen om ultraljudssensorn skulle detektera ett för nära avstånd till hindret. Utöver dessa pinnar så ska VCC, AVCC (matningsspänning +5V), GND (jord) och RESET kopplas in.



Figur 15: Kopplingsschema Sensor-AVR

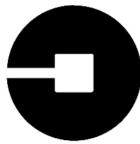
5.3 Komponenter

Denna sektion beskriver i detalj de tekniska specifikationerna på de olika valda komponenterna till sensormodulen.

- 1 ATmega1284P
- 1 Ultrafjärdssensor - SRF04
- 2 Halleffektsensorer - A1120
- 1 RPI Kamera

5.3.1 Mikroprocessor - ATmega1284P

Mikroprocessorn ATmega1284P har valts istället för ATmega16 då de 2 halveffektswitcharna på hjulen som ingår i sensormodulen lämpligen använder input capture för att fånga upp de pulser switcharna genererar. ATmega1284P har 2 16-bitars timers med separata pinnar för input capture jämfört med ATmega16 som har 1 16-bitars timer med input capture. Med detta val av mikroprocessor kan båda halveffektswitcharna kopplas in och behandlas separat.



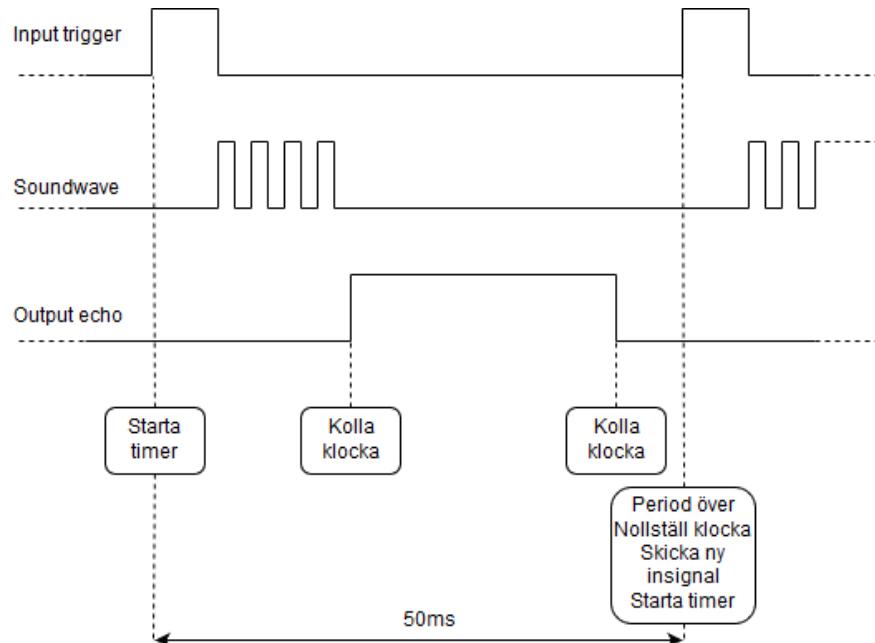
UNTER

5.3.2 Ultraljudssensor - SRF04

Ultraljudssensorn SRF04 är en avståndssensor som med hjälp av ultraljud mäter avstånd till objekt. Genom att skicka ut en ljudpuls med koniskform som färdas i ljudetshastighet för att sedan reflekteras tillbaka till sensorn kan avståndsdetektion genomföras. Sensorn kommer monteras fram på bilen på en höjd lämplig för att detektera de hinder som definierats i Banspecifikationen.

Sensorn startas genom att man skickar en input trigger till sensorn, detta kommer ske periodiskt med hjälp av PWM. Denna puls är en logisk etta som sätts hög i minst 10μ sekunder. Sensorn skickar då ut en 8-cyklistisk ultraljudspuls på 40 kHz och sätter då sin echo-output hög. Den lyssnar därefter efter ett eko och så fort den får respons av ett eko sätts echo-output till låg. Om inget detekteras innan 36 ms sätts echo-output till låg oavsett. Echo-output kan ses som en pulsbredd där bredden kan räknas som distansen till hindret. Efter echo-output går sensorn ner i vila i 10 ms för att sedan ha möjligheten att erhålla nästa input trigger.

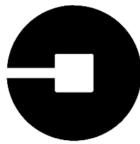
Händelseförfloppet visas i figur 16. En input trigger skickas och klockan startar då sin beräkning, när input trigger blir låg skickas en ljudvåg iväg. När sedan output echo får positiv flank kollar man klockan för att sedan kolla klockan ytterligare en gång då output echo får negativ flank. Intervallet mellan de båda tiderna jämförs och med hjälp av tidskillnaden kan en avståndsmätning genomföras. Efter att output echo fått negativ flank nollställs klockan och en ny input trigger kan skickas.



Figur 16: Händelseförflopp ultraljudssensor

Ekvation 4 beskriver valet av periodtid. Genom att sätta periodtiden till 50 ms försäkrar man sig om att hela sensorns utförande kommer att kunna ske innan nästa avståndssökning inträffar,

$$T = 10^{-6} + 2 \cdot 10^{-2} + 36 \cdot 10^{-6} + 10^{-3} \approx 48 \cdot 10^{-3} \quad (4)$$



UNTER

Med en periodtid på 50 ms erhåller man en frekvens på 20 Hz vilket innebär att man på en sekund gör 20 stycken avståndsmätningar. Skulle bilen exempelvis färdas i 1 m/s så innebär det att man erhåller en ny avståndsmätning var femte centimeter, vilket anses vara inom rimliga värden.

Ekvation 5 beskriver hur många systemklockpulser som krävs för att representera tiden för ultraljudssensorn. Låt säga att processorns klockfrekvens är $1.8432 \cdot 10^6$ Hz och med ultraljudssensorns frekvens på 20 Hz skulle det medföra att det antal systemklockpulser som krävs är 92159.

$$\text{Ticks} = \frac{f_{clock}}{f_{ultra}} - 1 = \frac{1.8432 \cdot 10^6}{20} - 1 = 92159 \quad (5)$$

Då ultraljudssensorn använder sig av en 8-bitars klocka för att beräkna tiden kan inte talet 12499 beräknas. En 8-bitars klocka kan enbart räkna upp till 255. Man behöver således utnyttja en prescaler. Ekvation 6 beskriver det värdet man behöver för att skala ner systemklockpulserna. Med valet $x = 361$ behöver man finna den prescaling faktorn på ATmega1284P som kan representera talet 361, vilket är en prescaling på 1024.

$$x = \frac{92159}{255} \approx 361 \quad (6)$$

Timern räknas med prescaling 1024 endast upp 1 gång var 1024:e systemklockpuls. Prescaling är en kompromiss mellan varaktighet och precision. Den prescaling som görs medför att ett event som i värsta fall sker på 1023:e systemklockpulsen kommer ha 8-bitars timern inställt på samma värde som om eventet skett på den första systemklockpulsen. Med en systemklocka på 1.8432 MHz fås det maximala möjliga felet i tidtagningen som

$$\text{Maxtimerfel} = \frac{1023}{1.8432 \cdot 10^6} \approx 0,000555\text{s} \quad (7)$$

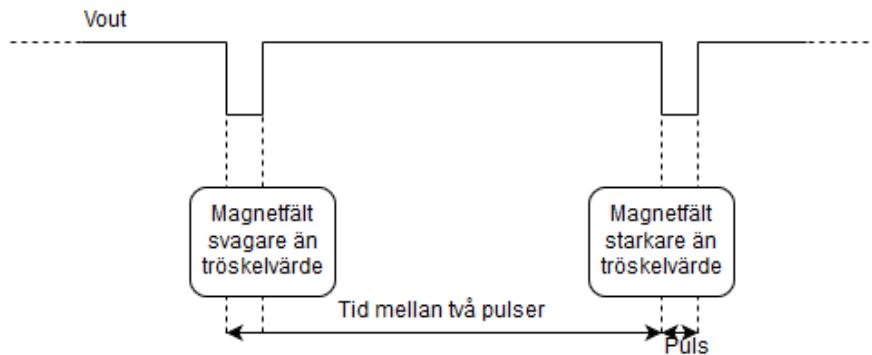
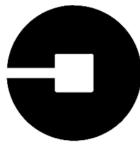
Om bilen antas färdas i den konstanta hastigheten 1m/s innebär denna osäkerhet på cirka 0.555ms beräknad i ekvation 7 en osäkerhet på erhållna mätvärden på cirka 0,555mm, se ekvation 8.

$$\text{Mätfel} = 1\text{m/s} * 0,000555\text{s} = 1\text{m/s} * 0,000555\text{s} = 0,000555\text{m} = 0,555\text{mm} \quad (8)$$

Ett mätfel på max 0,555mm anses vara försumbart för de avståndsberäkningar ultraljudssensorn kommer användas till. Prescaling 1024 bör alltså inte innehålla några större problem för sensorns funktion.

5.3.3 Halleffektswitch - A1120

Halleffektsswitchen A1120 är en switch som reagerar på ett extern magnetfält. Då det externa magnetfältet är ortogonal till halleffekten och starkare än ett tröskelvärde går signalen V_{out} från switchen låg. När det externa magnetfältet blir svagare än tröskelvärdet går signalen V_{out} hög igen, se figur 17. Tiden mellan två pulser antas vara kort nog att 16-bitars klockorna inte behöver prescalas men om detta inte visar sig vara fallet kommer en prescaling av klockorna utföras.



Figur 17: Händelseförlopp halleffektsswitch

Halleffektsswitchar finns på bilens bakhjul och används för att beräkna hur fort bilen färdas. Genom att koppla in V_{out} från de båda switcharna till vars in input capture pin på ATmega1284P kan tiden mellan erhållna pulser beräknas, ett JSP diagram för halleffektsswitcharna finns i figur 20. Att magneterna sitter jämnt fördelade på hjulet och med kännedom om hjulens radie kan det avståndet som färdades mellan två pulser samt hastigheten lätt beräknas, se ekvation 9 samt 10. Bilen kommer färdas i ganska låga hastigheter så felet från att hjul slirar anses försumbart.

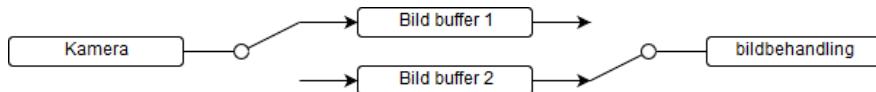
$$\text{Sträcka} = \frac{2 \cdot \text{hjulradie} \cdot \pi}{\text{antal magneter}} \quad (9)$$

$$\text{Hastighet} = \frac{\frac{2 \cdot \text{hjulradie} \cdot \pi}{\text{antal magneter}}}{\text{tid mellan pulser}} \quad (10)$$

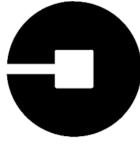
5.3.4 Kamera - RPI Camera

Kameran kommer vara en RPI kameramodul som kopplas in direkt till CSI porten på enkortsdatorn RPI. Kameran monteras längst fram på bilen nära vägen så att bilderna för att detektera körfält blir så bra som möjligt. Bilderna från kameran bildbehandlas i den del av sensormodulen som ligger på RPI:n för att detektera linjerna för körfälten och beräkna bilens nuvarande position på vägen.

Modulens datorseende använder sig av datorseende biblioteket OpenCV och kommer att kodas i programmeringsspråket C++. Biblioteket är uppbyggt av flera moduler och de moduler som krävs för bilens datorseende är core-modulen för att utföra beräkningar på bilden och imgproc-modulen för att utföra bildbehandlingen. Ett JSP diagram för datorseendet finns i figur 21. Datorseendet kommer att använda sig av två bildbuffrar, se figur 18, för att försäkra sig om att en bild som genomgår bildbehandling inte skrivs över av en ny bild från kameran innan bildbehandlingen är klar.



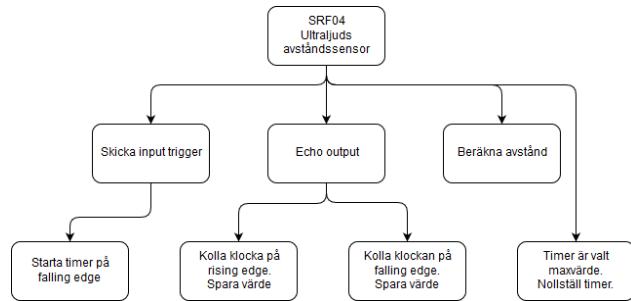
Figur 18: Tagna bilder sparas i två bildbuffrar



UNTER

5.4 JSP-diagram för avståndssensor

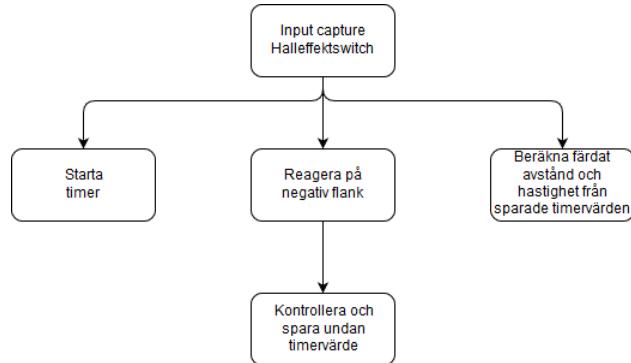
I figur 19 nedan beskrivs strukturen över hur programkoden för avståndssensorn kommer att se ut. Genom att beskriva programkoden med ett JSP-digaram blir det enkelt att se hur implementeringen av respektive funktion kommer att utföras.



Figur 19: JSP-diagram för avståndssensorn SRF04

5.5 JSP för HallEffektSwitch

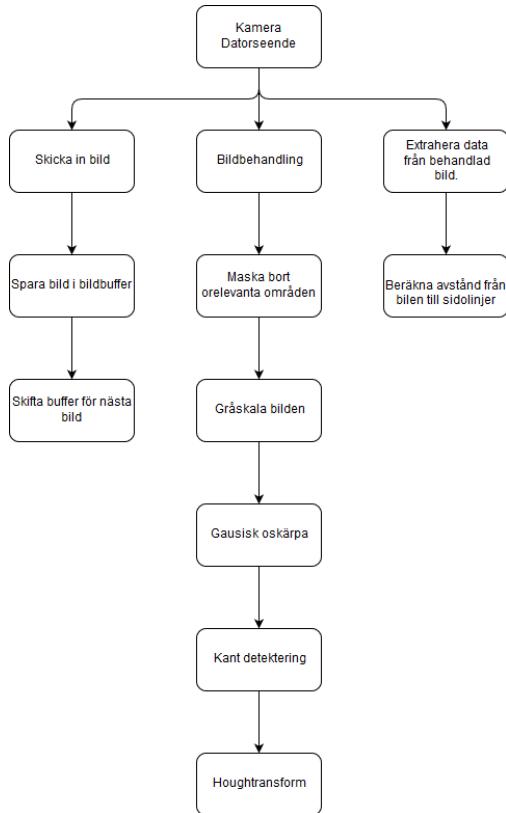
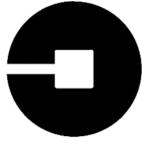
I figur 17 nedan beskrivs strukturen över hur programkoden för halleffektswitcharna kommer att se ut.



Figur 20: JSP-diagram för halleffektswitch

5.6 JSP-diagram för datorseendet

Då kameran kan anses vara bilens ögon är det kamerans uppgift att detektera var på vägen bilen befinner sig. Genom att med hjälp av datorseendet kan vägens vägkanter detekteras. Nedan i figur 21 beskrivs programkoden för hur vägens kantlinjer kommer att detekteras med hjälp av datorseende.



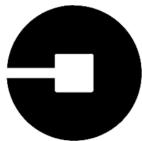
Figur 21: JSP-diagram för datorseende

6 PERSONDATOR

Denna sektion redogör för den externa persondatorn som används för att användaren ska kunna kommunicera med bilen trådlöst. Persondatorns uppgift i systemet är att skicka data till och ta emot data från bilen. Kommunikation med bilen sker trådlöst med hjälp av RPI:ns kommunikationsmodul. Persondatorn ska ta emot data, lagra datan och även skapa ett användarvänligt GUI.

6.1 Lagring av data

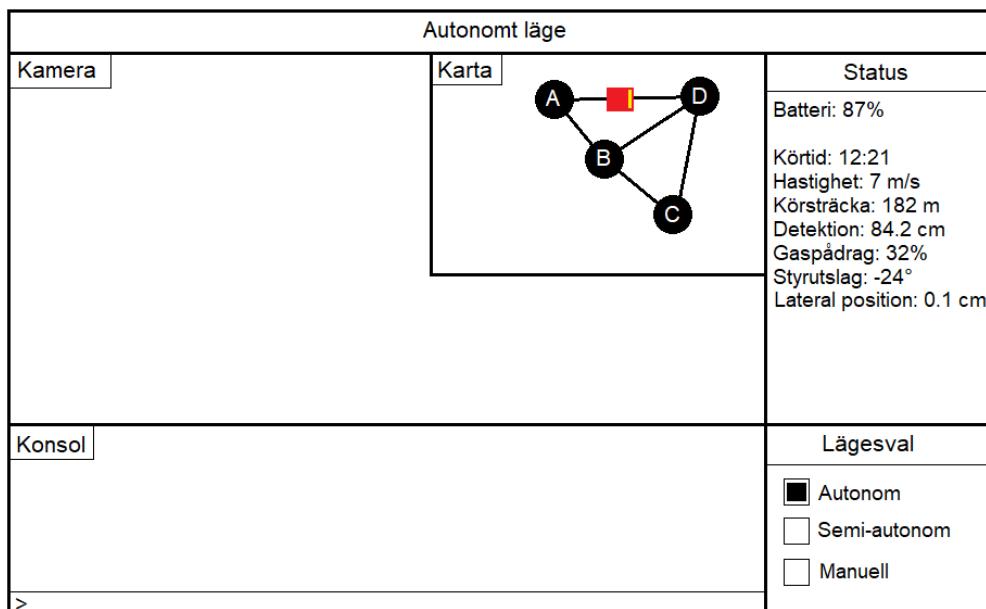
Lagring av data kommer ske med att persondatorn tar emot data och lagrar detta. Daten kommer att sparas i ett MATLAB-vänligt format. Den lagrade datan kommer därefter att kunna presenteras i MATLAB i form av grafer och vektorer.



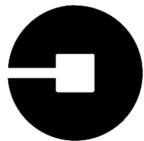
UNTER

6.2 Grafiskt gränssnitt

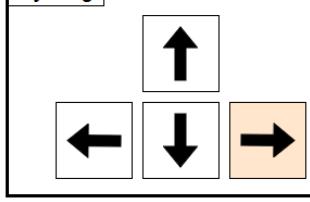
Denna sektion går igenom det tänkta grafiska gränssnittet som ska implementeras. Figur 22 och 23 visar hur GUI:t är tänkt att se ut. GUI:t kommer att skapas i Python och använda sig av Tkinter graphical user package. Detta då Python är ett enkelt verktyg för att implementera grafiska gränssnitt. Tkinter-biblioteket för Python är de facto standard för det grafiska gränssnittet, eftersom det är så väl använt finns det mycket information att finna vilket bör leda till en smärtfri implementering av GUI:t.



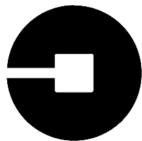
Figur 22: Skiss av användargränssnittet i autonomt läge.



UNTER

Manuellt läge		
Kamera	Styrning	Status
		Batteri: 87% Körtid: 12:21 Hastighet: 7 m/s Körsträcka: 182 m Detektion: 84.2 cm Gaspedal: 32% Styrtslag: -24°
Konsol	Lägesval	
>	<input type="checkbox"/> Autonom <input type="checkbox"/> Semi-autonom <input checked="" type="checkbox"/> Manuell	

Figur 23: Skiss av användargränssnittet i manuellt läge.

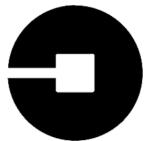


UNTER

REFERENSER

- [1] Raspberry Pi Foundation, “Raspberry Pi 3 Model B,” <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, 2019, [Online; accessed March 5, 2019].
- [2] Future Technology Devices International Ltd., “TTL-234X Serial Cables,” <https://www.ftdichip.com/Products/Cables/TTL234XSerial.htm>, 2019, [Online; accessed March 5, 2019].
- [3] Raspberry Pi Foundation, “Setting up a Raspberry Pi as an access point in a standalone network (NAT),” <https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md>, 2019, [Online; accessed March 5, 2019].
- [4] ——, “SSH (Secure Shell),” <https://www.raspberrypi.org/documentation/remote-access/ssh/README.md>, 2019, [Online; accessed March 6, 2019].

7 APPENDIX



Autonom taxibil

2021–07–09

UNTER

F FÖRSTUDIE KOMMUNIKATION

Kommunikation

4 juni 2019

Förstudie: Kommunikation

Filip Jaredson, Nicholas Sepp

4 juni 2019

Version 1.1

Status

Granskad	FJ, NS	4 juni 2019
Godkänd		

Projektidentitet

Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Dokumentförfattare

Namn	Grupp	E-post
Filip Jaredson	8	filja057@student.liu.se
Nicholas Sepp	9	nicse725@student.liu.se

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförd av	Granskad
0.1	2019-03-04	Första utkast	FJ, NS	FJ, NS
1.0	2019-03-27	Första version	FJ, NS	FJ, NS
1.1	2019-04-15	Korrigerad version	FJ, NS	FJ, NS

INNEHÅLL

1	Inledning	1
1.1	Sammanhang	1
1.2	Syfte och mål	1
1.3	Problemformulering	1
1.4	Metod	1
2	Trådlös kommunikation	2
2.1	WLAN och Wi-Fi	2
2.2	Bluetooth	4
3	Bussar	7
3.1	Universal asynchronous receiver-transmitter (UART)	7
3.2	Serial Peripheral Interface (SPI)	8
3.3	Inter-Integrated Circuit (I2C)	9
4	Resultat	10
4.1	Trådlös kommunikation	10
4.2	Bussar	10
5	Diskussioner och slutsatser	11
5.1	Trådlös kommunikation	11
5.2	Bussar	11
6	Referenser	12

1 INLEDNING

1.1 Sammanhang

Denna förstudie utförs inför det kandidatprojekt i elektronik som ska genomföras. Projektet består av att utveckla en autonom taxibil från idé till färdig produkt. Systemet kommer bestå av ett bilchassi med flera olika datorer på och en central styrenhet som ska kommunicera med varandra. Bilens datorer kommer vara två stycken AVR-processorer och en Raspberry Pi 3 Model B som kommer behöva kommunicera sinsemellan. Dessa datorer ska även kunna kommunicera trådlöst med den centrala styrenheten.

1.2 Syfte och mål

Syftet med denna förstudie är att samla ett gediget kunskapsunderlag som kan utnyttjas inom det projekt som ska utföras. Det finns flertalet olika lösningar på de problem som tillhör utvecklingen av systemet och att bestämma den optimala lösningen till varje delproblem kräver gedigen kunskap och erfarenhet. På grund av trögheten som finns i ett stort projekt med många medlemmar och mycket resurser är det av stor vikt att redan i den tidiga planeringsfasen bestämma vilka lösningar och teknologier som lämpar sig bäst för systemet. En illa vald lösning utnyttjar mer resurser i både material och tid än en bättre vald lösning och i vissa fall kan inte en dåligt vald lösning åtgärdas inom projektets tidsram.

1.3 Problemformulering

I denna förstudie kommer två sammanhängande kommunikationsproblem att undersökas med viss översiktlighet.

- Bilen kommer behöva kommunicera trådlöst med en extern styrenhet, i det vanliga fallet en bärbar personadator. Via denna trådlösa kommunikation ska bland annat manuella styrkommandon och autonoma köruppdrag kunna skickas från datorn till bilen och från bilen till datorn ska det bland annat skickas data som körtid och körsträcka. Vilken kommunikationsteknologi lämpar sig för systemet? Kommunikationens säkerhet, robusthet, räckvidd, störtålighet och dataöverföringshastighet är i detta fall intressanta parametrar att jämföra mellan de olika alternativen.
- De olika processorerna på bilen kommer behöva kommunicera sinsemellan med hjälp av bussar. Olika bussar har olika förutsättningar och möjligheter, i denna förstudie kommer främst bussteknologiernas dataöverföringshastighet, robusthet och kompatibilitet vara av intresse.

1.4 Metod

Denna förstudie utförs genom att samla relevant information med avseende på problemformuleringarna till ett kunskapsunderlag vilket sedan används för att besvara Kunskapsunderlaget hämtas från diverse olika källor men som primära källor kommer akademiska artiklar, protokollspezifikationer och böcker användas. Dokumentation på exempel av implementationer av teknologierna som undersöks utnyttjas även för att redogöra för teknologiernas möjligheter och begränsningar. Med detta gedigna kunskapsunderlag ska olika teknologier jämföras på intressanta parametrar så att slutsatser kan dras om hur problemformuleringarna bör besvaras.

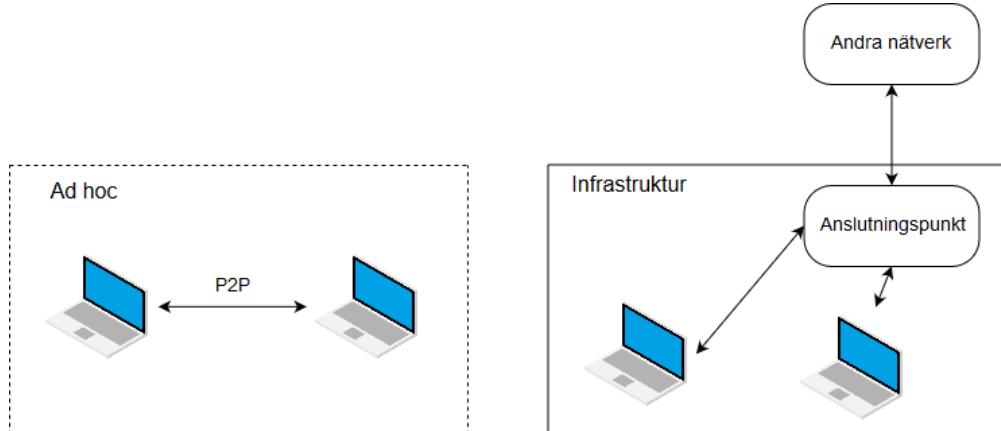
2 TRÅDLÖS KOMMUNIKATION

Följande avsnitt redogör för två olika trådlösa kommunikationsteknologier, nämligen Wi-Fi och Bluetooth.

2.1 WLAN och Wi-Fi

WLAN (Wireless Local Area Network) är ett samlingsnamn för trådlösa kommunikationsteknologier som kopplar ihop två eller flera enheter för att skapa ett lokalt nätverk (LAN) i ett begränsat område som ett hem, en kontorsbyggnad etc. Den stora fördelen med ett WLAN jämfört med ett LAN som etableras med ethernet-kablar är att enheterna (och därmed användarna) kan röra sig i ett WLAN så länge de är inom nätverkets trådlösa räckvidd. Via, oftast, ett modem kan ett WLAN kopplas till vidare Internet. Den vanligaste typen av WLAN i modernt bruk är IEEE 802.11-familjen av standarder som marknadsförs under namnet Wi-Fi [1]. I denna förstudie ligger fokus på IEEE 802.11-protokollen.

Enligt Prasad finns det två olika typer eller modeller av WLAN; teknologin som tillhandahåller möjligheten att bilda kopplingar i en infrastruktur och teknologin som tillhandahåller möjligheten att bilda kopplingar från en enhet till en annan (ad hoc nätverk) [2]. Teknologin för ad hoc nätverk kommer i denna förstudie vara av intresse då uppkopplingen mellan systemets externa dator och bilen ingår i ett sådant nätverk. I ett ad hoc nätverk kommunicerar enheterna med varandra peer-to-peer (P2P) medan i ett nätverk i en infrastruktur kommunicerar enheterna till en trådlös anslutningspunkt som kan agera som en brygga till andra nätverk (t.ex. internet), se figur 1. WLAN ersätter inte lösningar som utnyttjar kablar men snarare komplementerar dem; WLANs erbjuder möjligheten för att upprätta nätverk i miljöer där det är svårt att etablera nätverk med kablar. WLANs ger en utökad flexibilitet till nätverket; det blir lättare att flytta och utvidga nätverk samt att enheter i rörelse kan vara fortsatt uppkopplade.



Figur 1: Översiktligt figur över WLAN:s två olika lägen.

2.1.1 IEEE 802.11 standarderna

De tidigaste trådlösa kommunikationsteknologierna uppfanns 1990 och utnyttjade 900 MHz frekvensbandet. Dessa teknologier kunde uppnå 1 Mb/s i hastighet, jämförvis kunde LAN under samma tid uppnå hastigheter på 10 Mb/s. Dessa tidiga implementationer följe ingen standard. Projektet inom IEEE som utvecklade 802.11-protokollet påbörjades 1990 och hade som syfte att utveckla standarder för MAC (Medium Access Control) och PHY (Physical) för

fasta och rörliga trådlösa enheter [1]. Det ursprungliga 802.11 protokollet godkändes 1997 och är 802.11 familjens grundprotokoll. Sedan grundprotokollet publicerades har förbättringar tidvis introducerats som främst förbättrat prestanda och säkerhet. Dessa förbättringar till grundprotokollet publiceras som tillägg, som identifieras av en gemen bokstav. Ett exempel på ett sådant tillägg är 802.11b som uppstod 1999 och opererar i samma 2.4 GHz frekvensband som det ursprungliga protokollet men med förbättrad dataöverföring; det ursprungliga 802.11-protokollet har stöd för dataöverföringshastigheter på 1 och 2 Mb/s medan 802.11b har stöd för hastigheter på 5.5 och 11 Mb/s. 802.11b blev populärt väldigt fort.

2.1.2 Raspberry Pi 3 Model B

I projektet används en mikrodator av typ Raspberry Pi 3 Model B. Denna mikrodator har en Broadcom BCM43438 som sin nätverksmodul[3][4]. Denna nätverksmodul tillhandahåller både WiFi 802.11a/b/g/n och Bluetooth Low Energy (BLE) 4.0 som möjliggör trådlös kommunikation. 802.11n protokollet specificerar att kommunikation ska kunna ske på både 2.4 och 5 Ghz frekvensbanden där det senare frekvensbandet dock inte behöver implementeras. I mikrodatorn har detta protokoll implementerats så att endast 2.4 Ghz frekvensbandet kan användas[5].

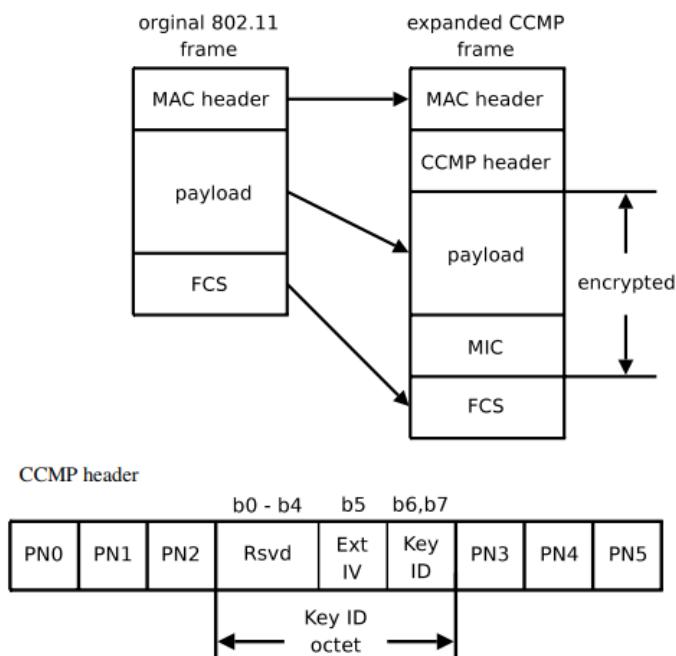
802.11n-protokollet ska kunna nå dataöverföringshastigheter från 54 till 600 Mb/s och har en approximativ räckvidd på 70 meter i inomhusmiljöer och 250 meter i utomhusmiljöer. Systemet som utvecklas ska operera inomhus men vilken räckvidd som kan uppnås kan variera beroende på störningarna som systemet utsätts för; till exempel sitter mikrodatorn monterad på den rörliga bilen, vilket kan påverka räckvidden. 802.11n var protokollet som introducerade multipla antenner och MIMO (Multiple-Input and Multiple-Output) vilket delvis förklarar den stora ökningen av netto dataöverföringshastighet från tidigare protokoll, från 54 till uppemot 600 Mb/s, vilket uppnås bland annat genom att använda 4 dataströmmar som var har en bandbredd på 40 MHz. Säkerhetsprotokollet som används för 802.11n är Wi-Fi Protected Access II (WPA2). Denna certifiering inkluderar ett stöd för CCMP, som är ett enkryptionsprotokoll som baseras på AES och introducerades med 802.11i tillägget till standarden. Enligt specifikation måste detta enkryptionsprotokoll implementeras för att 802.11n ska uppnå höga dataöverföringshastigheter; om protokollet inte används kommer inte hastigheten att överstiga 54 Mb/s.

2.1.3 CCMP

Counter Mode Cipher Block Chaining Message Authentication Code Protocol eller CCM mode Protocol (CCMP) baseras på Advanced Encryption Standard (AES)[1][6]. CCMP tillhandahåller starka kryptografiska metoder och rekommenderas att användas istället för Temporal Key Integrity Protocol (TKIP) i 802.11i-tillägget. AES är ett mycket starkare enkryptionsprotokoll än RC4 som används i TKIP. TKIP existerar fortsatt i protokolltillägget endast för att enheter som implementerar detta ska vara bakåtkompatibla med enheter som opererar på tidigare versioner av 802.11-protokollet.

En översikt av CCMPs funktion visas i figur 2. 802.11 MAC-ramar skickas till CCMP för behandling. Dessa ramar består av en MAC-header, data i klartext (payload) och en sekvens för ramkontroll (Frame Check Sequence, FCS). CCMP tillhandahåller en integritetskod för meddelanden (Message Integrity Code, MIC) som tillhandahåller autentisering och integritet för den skickade payload datan. MIC och payload datan enkrypteras tillsammans i den nya expanderade CCMP-ramen. En CCMP-header läggs till den ursprungliga MAC-headern. FCS läggs även till i CCMP-ramen. Denna behandling visas i övre delen av figur 2. Nedre delen av figur 2 visar CCMP-headerns fält. CCMP-headern består av en ExtIV, en KeyID och ett paketnummer (Packet Number, PN). PN0 är den minst signifikantha biten. CCMP genererar en ny temporär nyckel för varje session och ett nonce-värde för varje ram. Nonce-värdet är en 48 bitars PN som är enkrypterad med den temporära nyckeln. Standarden varnar för att säkerheten kan hotas om PN

återvänts med samma temporära nyckel. PN inkrementeras därför för varje ram. CCMPs inkapslingsprocess beskrivs som följer. Den övriga autentiseringsdatan (Additional Authentication Data, AAD) konstrueras av fält från MPDU-headern. PN, det andra adressfältet (A2) och prioritetsfältet i MAC-headern används för att konstruera nonce-värde. PN och KeyID placeras i CCMP-headern. Sammanslagningen av det ursprungliga payload datan från MAC-ramen och MIC enkrypteras med AES genom att använda den temporära nyckeln, AAD och nonce-värde. Dekryptionsprocessen beskrivs som följer. Fält från MAC- och CCMP-headern extraheras för att konstruera AAD och nonce-värde. Den krypterade payload datan dekrypteras för att producera datan i klartext och MIC:n. Den ursprungliga ramen återskapas genom att slå samman den ursprungliga datan i klartext med MAC-headern.



Figur 2: Utvidgad CCMP-ram och CCMP-header. [1], s. 108

2.2 Bluetooth

Bluetooth är en trådlös kommunikationsteknologi som standardiseras av IEEE som IEEE 802.15.1 men idag tillhandahålls av Bluetooth Special Interest Group (SIG). I grundspezifikationen för Bluetooth 5.1, den senaste versionen, skriver Bluetooth SIG att Bluetooth är en trådlös kommunikationsteknologi med kort räckvidd vilken är avsedd att ersätta kablar som kopplar ihop portabla och/eller fasta elektroniska enheter. Det finns två typer av Bluetooth system: Basic Rate (BR) och Low Energy (LE). LE systemet är designat att erbjuda trådlösa lösningar till produkter som kräver lägre strömförsljörning, komplexitet, dataöverföringshastighet, pulskvot och kostnad än BR. Då Bluetooth Low Energy (BLE) 4.0 tillhandahålls av Raspberry Pi:n som används i detta projekt kommer denna version av Bluetooth vara av störst intresse i denna förstudie[7][8].

Författarna till boken *Bluetooth Security* [9] skriver att Bluetooth utvecklades med en stor tyngd på kostnad och energiförbrukning, då Bluetooth ofta var tänkt att användas i portabla, batteridrivna enheter. Därmed har en avvägning mellan kostnad och energiförbrukning på ena hand och allmän prestanda på andra hand gjorts under utveckling. Till exempel är kraven på radiokänslighet så låga att implementationer kan göras billigare, där priset som betalas är en sämre räckvidd. Dock är ett huvudmål i designen att systemet ska vara robust i miljöer med brus, detta eftersom störningar anses vara mer av en begränsande faktor för den uppfattade prestandan än vad räckvidd är. Författarna skriver även att Bluetooth är väl lämpat för ad hoc-nätverk: Bluethooths flexibla master-slave koncept introducerades för att fungera väl med ett föränderligt system av kommunicerande enheter. På grund av de flerfaldiga kraven på typer av datatrafik som flera olika användningsområden ställer på kommunikationen, kan Bluetooth hantera ett flertal dataöverföringskanaler: asynkrona, isokrona och synkrona kanaler tillhandahålls. Det är även möjligt för en enhet att använda asynkron (data) och synkron (ljud) överföring samtidigt.

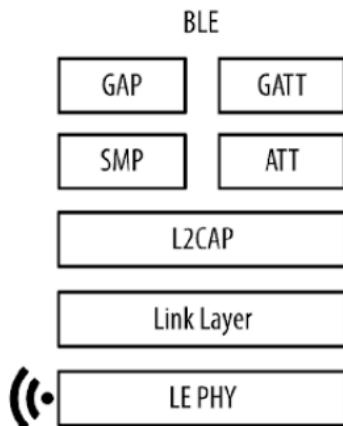
2.2.1 **Bluetooth Low Energy**

Enligt författarna till *Getting Started with Bluetooth Low Energy* [10] kan BLE-teknologins snabba inträde till vida industriavändning delvis förklaras av teknologins designfokus på låg energiförbrukning och resurseffektivitet. Detta i kombination med den explosiva tillväxten av portabla datorenheter som smartphones och surfplattor haft sedan introduktionen av BLE året 2010 och att industrijättar som Apple och Samsung tidigt och aktivt använde BLE ledde till en vidare adoption av teknologin. Författarna skriver att BLE på många sätt var rätt teknologi med de rätta kompromiserna vid rätt tid.

BLE 4.0 opererar på 2.4 GHz ISM frekvensbandet och använder sig av frekvenshopande mottagare för att motverkar störningar. En dataöverföringshastighet på 1 Mb/s stöds och den teoretiska maximala räckvidden ligger på 100 m, dock är 30 m en vanligtvis pålitlig räckvidd. 2 till 5 meter är ett typiskt operationsomfång i många implementeringar, vilket ofta är ett medvetet val för att maximera batteritid utan att avståndsbegränsningen skulle vara ett problem för slutanvändaren. För BLE-enheter med radiosändare har en maximal uteffekt på 10 mW (+10 dBm) specificerats. Två multipla åtkomstscheman nyttjas: Frequency division multiple access (FDMA) och time division multiple access (TDMA). 40 fysiska kanaler var separerade av 2 MHz används i FDMA schemat. 3 används som annonseringskanaler och resterande 37 används som datakanaler. Ett pollingsschema som baseras på TDMA används då en enhet överför ett paket vid en förbestämd tid och en motsvarande enhet svarar med ett paket efter ett förbestämt intervall. Den fysiska kanalen är uppdelad i tidsenheter som kallas händelser. Data överförs mellan BLE enheter i paket som är placerade i dessa händelser. Det finns två typer av händelser: Annonserings- och uppkopplingshändelser, vilka bland annat används för att koppla master med slavar.

En översikt av protokollstacken för BLE ges i figur 3 och i detta stycke kommer stackens olika delar att redogöras för översiktligt. Ovanför den fysiska kanalen (LE PHY) finns länkar, kanaler och associerade kontrollprotokoll. Hierarkin är som följer: fysisk kanal, fysisk länk, logisk transport, logisk länk och L2CAP kanal. I en fysisk kanal bildas en fysisk länk mellan master och varje slav. Fysiska länkar slavar sinsemellan i ett BLE-nätverk, som kallas piconet i specifikationen, stöds inte. Slavar tillåts inte att ha fysiska länkar till mer än en master vid varje tid. Rollbyte mellan master och slav är inte heller tillåtet. Detta betyder att i varje piconet finns det bara en master och att alla slavar har en fysisk länk till denna master. Den fysiska länken används som transportkanal för en eller flera logiska länkar som stödjer asynkron trafik. Ett kontrollprotokoll för länklagren och de fysiska lagren bär över logiska länkar utöver användardata. Detta är länkkörprotokollet (link layer protocol, LL). Enheter som är aktiva i ett piconet har en default LE asynkron uppkoppling logisk transport (LE asynchronous connection logical transport, LE ACL) som används för att transportera LL protokollsinalerna. Link Layer-funktionen använder LL protokollet för att kontrollera opera-

tionerna hos enheterna som är uppkopplade till piconetet och tillhandahåller tjänster för att handskas med de lägre arkitektoniska lagrarna (PHY och LL). Ovanför länklagret tillhandahåller L2CAP-lagret en kanalbaserad abstraktion för applikationer och tjänster. Ovanför detta lager finns ytterligare lager, två av dessa är säkerhetshanteringsprotokollet (Security Manager Protocol, SMP) som använder en fix L2CAP-kanal för att implementera säkerhetsfunktionalitet mellan enheterna och attributprotokollet (Attribute protocol, ATT) som tillhandahåller en metod för kommunikation av små mängder data över en fix L2CAP-kanal. Det sistnämnda protokollet används även av enheter för att avgöra tjänsterna och kapabiliteterna hos andra enheter. I det översta lagret i figuren finns det generiska attributprofilen (Generic Attribute Profile, GATT) som specificerar de allmänna profilkav och den generiska åtkomstprofilen (Generic Access Profile, GAP). Detta är den minimala implementationen av BLE.



Figur 3: Översiktligt figur av protokollstacken för BLE. [10], s. 4

Länklagret (LL) tillhandahåller enkryptions och verifiering med Counter with Cipher Block Chaining Message Authentication Code (CCM) Mode, som ska vara implementerat i enlighet med IETF RFC 3610 och AES-128 blockenkryptering.

3 BUSSAR

För kommunikation mellan mikrodatorer brukar olika specifikationer användas för att överföra information mellan enheterna på effektivast sätt. För att ge en bakgrund och beskrivning av tillämpningsfall för kommunikationssätten är det viktigt att undersöka busskommunikation såsom UART, SPI och I2C.

3.1 Universal asynchronous receiver-transmitter (UART)

UART är ett asynkront, peer-to-peer interface, vilket innebär att kommunikation över UART sker mellan två enheter utan någon central klocka. Det används flitigt i industrin, på grund av dess mångsidighet och enkelhet. UART-protokollet bygger på input/output shift registers, som skiftar in nya informationen och fasar ut gammal information. En vanlig UART brukar även innehålla en klocka, inställd på någon multipel av bitraten för att bussen ska kunna läsa information mitt i en bit-period, transmit/receive-control och read/write-control. Övrig funktionalitet som kan finnas implementerad är FIFO-buss-minnen, transmit/receive-buffrar och databuffer till systembussen.

3.1.1 Installation

Att installera UART mellan två enheter är relativt simpelt. Det består främst i att konfigurera båda enheterna att arbeta med samma baud rate och datastruktur. En hög baud rate innebär ett stort antal förändringar i bitar över tid, medan en låg baud rate betyder att data skickas saktare. Om enheternas baud rate inte är synkad finns det risk för att ena enheten förväntar sig information när den andra enheten inte har skickat den än, och kan på så sätt misstolka stoppbitar för att avbryta inkompletta meddelanden. Hur datastrukturen kan se finnas beskrivet i figur 4 och det viktiga när man konfigurerar det är att antalet start-, data-, parity- och stoppbitar konfigureras till samma värde hos både sändare och mottagare.

3.1.2 Prestanda

UARTs protokoll består i en startbit, mellan fem och åtta databitar, möjligtvis en parity-bit och slutligen en eller två stoppbitar. Parity-biten finns till som en säkerhetsåtgärd för att kontrollera att datan som kommit fram stämmer genom att parity-biten bekräftar om den är jämn eller udda, beroende på inställningarna. Stoppbiten eller -bitarna återställer bussen till viloläge, och efter de har skickats kan ett nytt paket skickas. Två stoppbitar innebär att bussen måste vara i viloläge en bit längre innan den kan skicka paket igen.



Figur 4: Diagram över UARTs dataram i det fall då 7 databitar, 0 parity-bitar och 1 stoppbit används. Källa: [11]

Överföringshastigheten i UART avgörs av baud raten som sätts när man konfigurerar porten. Högre hastighet innebär att fler databitar kommer skickas i sekunden, men även ökad risk för störningar eller förlorad data.

3.1.3 Tillämpning

Fram till början av 2000-talet var standarden RS-232 den främsta kanalen för kommunikation över UART. Därefter började de externa RS-232-portarna ersättas med USB-portar, eftersom USB hade högre överföringshastighet och längre spänningssving. I dagsläget används RS-232 fortfarande ibland i industrimaskiner och vid nätverkskommunikation där prestanda inte är prioriterat, på grund av dess enkelhet, men i allmänhet är moderna datorer inte utrustade med RS-232-portar. Eftersom UART endast kan kommunicera mellan två enheter är tillämpningen vanligast i system bestående av två enheter, men det är även möjligt att tillämpa en UART-buss mellan flera enheter.

Bussen lämpas bra som utgångspunkt för tvåvägskommunikation i allmänhet, på grund av att kommunikationen över bussen är billig och lättimplementerad. Exempelvis passar UART bra för kommunikation med sensorer som ska kommunicera med en mikrokontroller, eller om ett system vill kunna skicka data med liten belastning på datorn.

3.2 Serial Peripheral Interface (SPI)

SPI är ett synkront seriellt kommunikationsinterface som används för att kommunicera mellan flera enheter på samma buss. Kommunikation över en SPI-buss sker genom en master-slave-struktur, där en master styr när och hur kommunikation sker över bussen. Kommunikation över SPI-bussen sker efter mastern pekat ut den slav som styrs genom att sätta biten för slave select (SS) till ett värde motsvarande den slaven, och möjligtvis vänta ut klockcykeln innan den skickar vidare info beroende på konfigurationen.

3.2.1 Installation

Installation av SPI är inte så mycket svårare än för UART. Man behöver i grund och botten konfigurera inställningarna för kommunikationen, till exempel om data ska tas emot på stigande eller fallande flank eller om klockans startläge är 0 eller 1. Storleken på bitar behöver inte konfigureras upp utan man kan, i teorin, skicka oändligt stora datamängder på en överföring. En annan egenskap hos SPI är att det är synkront, som innebär att det finns en gemensam SPI-klocka mellan enheterna för att synkronisera slavarna till mastern. Dataöverföringen i sig sker över en master in slave out (MISO)-port och en master out slave in (MOSI)-port, som skickar dataströmmen mellan de olika enheterna.

3.2.2 Prestanda

En typisk dataframe i en SPI-buss är strukturerad så mastern inledningsvis skickar en instruktionsbyte som avgör ifall slaven ska läsa eller skriva till bussen, följt av en destinationsadress att läsa från eller skriva till, en ström av data och slutligen stoppbit. Dataframes kan därmed bli väldigt stora i SPI-bussar utan att behöva delas upp över flera paket och ökar på så sätt effektiviteten hos bussen när mycket data ska överföras.

Dataöverföring över en SPI-buss sker med höga hastigheter. Det finns ingen definierad hastighetsgränsning, men har möjlighet att överföra data i över 10 Mb/s [12]. I samband med effektiviteten hos protokollets dataframes ger det SPI-protokollets höga prestanda.

3.2.3 Tillämpning

Då SPI har kapacitet att kommunicera relativt simpelt mellan flera enheter i ett system kan en SPI-buss användas när det finns flera enheter som delar resurser. Det brukar även tillämpas i de fall man ska skicka olika datatyper över bussen, eftersom användaren själv kan definiera en stoppbit och därefter skicka information av godtycklig storlek.

3.3 Inter-Integrated Circuit (I2C)

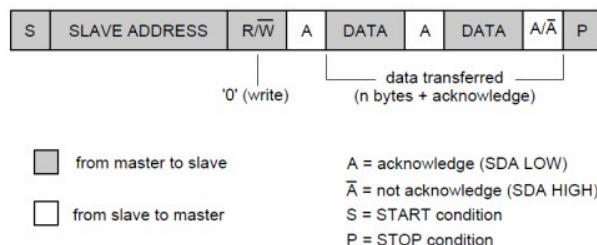
Slutligen finns ett protokoll som heter I2C. I2C är, likt SPI, ett master-slave-protokoll över en buss, men till skillnad från SPI kan en I2C-buss bestå av upp till 128 enheter med stöd för flera master-enheter. Godtyckliga enheter kan därmed kopplas in eller ersättas på bussen med minimal konfiguration. Protokollet i sig är väldigt flexibelt då det inte kräver någon konfiguration från enheternas håll, utan endast behöver interпретeras av den mottagande enheten.

3.3.1 Installation

Det inte minimala förberedelser när man installerar en I2C-buss. Bussen består i två kanaler, en till gemensamma klockan och en till skrivning av data. Är dessa inkopplade på rätt sätt bör inget externt konfigureras för att få systemet att fungera.

3.3.2 Prestanda

I2C använder sig av större, mer komplicerade dataframes och protokoll än både SPI och UART eftersom informationens destination inte är förutbestämd. Protokollet börjar med ett starttillstånd, följt av en kontrollbyte med adressen till målenheten och en read/write-bit som avgör om man är ute efter att läsa eller skriva från enheten. Därefter väntar sändaren från en acknowledge-bit för att bekräfta att mottagaren är redo för kommunikation. När acknowledge-biten tagits emot från mottagaren kan data börja överföras, och först skickas måladressen som delas upp på 7 bitars msb, vänta på acknowledge-bit från mottagare, och sedan 8 bitars lsb följt av en acknowledge-bit. Information som följer det varierar beroende på om målet är att skriva till eller läsa från måladressen. Ska informationen skrivas skickas datan följt av en acknowledge-bit och slutligen ett stopptillstånd. Efter ett stopptillstånd tagits emot vilar enheten tills den får ett nytt starttillstånd. Ska man läsa väntar istället målenheten på en ny kontrollbyte (inklusive starttillstånd) som slutar i 1, och efter den tagit emot det svarar den med en acknowledge-bit, läser data från måladressen, och svarar med en nacknowledge-bit (motsatt tillstånd från acknowledge) innan stopptillståndet.



Figur 5: Dataram för I2C-protokollet i det fall då en master vill skriva till en slav. Källa: [13]

Bandbreddmässigt kräver detta protokoll mer prestanda än de övriga då den behöver skicka mer bitar för en byte data, men trots det har I2C:s Ultra-Fast mode från 2012 lyckats möta 5 Mbit dataöverföring i sekunden.

3.3.3 Tillämpning

På grund av multimasterstrukturen hos protokollet passar I2C bra i stora nätverk, då central styrning kan bli svårt och I2C är decentraliserat. Det passar även bra i system där budget prioriteras över prestandan, då förlusten i hastighet kommer med fördelen att det bara krävs två pins för att kommunicera mellan enheter.

4 RESULTAT

I denna sektion kommer intressanta parametrar för olika kommunikationsprotokoll att jämföras. Vilka parametrar som är av intresse presenterades i problemformuleringen, se underrubriken [1.3](#). För de olika typerna av trådlös kommunikation ska säkerheten, robustheten och dataöverföringshastigheten att jämföras. För de olika typerna av busskommunikation ska dataöverföringshastigheten, robustheten och kompatibiliteten jämföras. Dessa jämförelser ska utvärderas och ligga som underlag för slutsatser om vilka tekniska lösningar är lämpligast för systemet som ska utvecklas.

4.1 Trådlös kommunikation

De två trådlösa kommunikationsprotokoll som är relevanta och ska jämföras är 802.11n-protokollet och Bluetooth Low Energy (BLE) 4.0, eftersom båda dessa protokoll tillhandahålls av nätverksmodulen på Raspberry Pi-enheten som används i systemet.

4.1.1 Säkerhet

Både 802.11n och BLE utnyttjar enkryptionsprotokollet CCM och är därmed jämförbara säkerhetsmässigt. CCM är ett kraftfullt enkryptionsprotokoll och tillhandahåller tillräcklig autentisering och integritet för de flesta privata trådlösa nätverk.

4.1.2 Robusthet & störningskänslighet

802.11n i detta fall och BLE i allmänhet opererar på 2.4 Ghz frekvensbandet vilket betyder att båda protokollens kommunikation kan störas av andra signaler på det frekvensbandet. En vanlig källa av störningar på detta frekvensband är många typer av mikrovågsugnar, detta eftersom mikrovågor har en våglängd som gör att de ockuperar det frekvensbandet.

4.1.3 Räckvidd & bandbredd

På grund av BLE:s stora designfokus på att användas i portabla enheter och vara resurseffektivt brukar kommunikationens räckvidd och dataöverföringshastighet att begränsas, främst i syftet att maximera enhetens batteritid. 802.11n har inte samma designmål och stödjer större både teoretiska och praktiska räckvidder och dataöverföringshastigheter. BLE har en pålitlig räckvidd på 30 meter, i många batterisnåla implementationer tenderar dock räckvidden vara runt 2 till 5 meter. 802.11n har en pålitlig räckvidd på 70 meter i inomhusmiljö. BLE stödjer en dataöverföringshastighet på 1 Mb/s, 802.11n stödjer dataöverföringshastigheter från 54 till 600 Mb/s.

4.2 Bussar

De protokoll som ska jämföras här är de som tidigare diskuterats i rapporten, det vill säga UART, SPI och I2C, eftersom alla tre protokoll stöds av Raspberry Pi-enheten i systemet.

4.2.1 *Prestanda*

I projektet kommer det vara viktigt att prestandan på bussarna som används är tillräckligt bra, eftersom det kommer ske mycket kommunikation på dem. Av de alternativ som undersökts i rapporten kan SPI leverera högst hastighet med stor bandbredd på dataöverföringen jämfört med UART och I2C. SPI kan, som tidigare nämnt, komma upp i 10+ Mbit/s [12], medan I2C kan skicka i 3.4 Mbit/s i High Speed Mode [14] och UARTs maximum standard baud rate är 115.2Kbit/s [15].

4.2.2 *Kostnad*

Då beställaren inte sätter budgetkrav på bilen är kostnad inget stort bekymmer för projektet, men från ett vetenskapligt perspektiv är det en användbar sak att undersöka. När det gäller kommunikationsbussarna som har diskuterats tidigare i rapporten ligger både UART och I2C väldigt bra till budgetmässigt gentemot SPI, eftersom de endast behöver två pins jämfört med SPI:s fyra pins.

4.2.3 *Simplicitet*

Slutligen är en av de kanske viktigaste aspekterna hos bussarna hur pass enkla de är att implementera. Simplaste protokollet av de som undersökts är UART, eftersom den kan skicka och ta emot information utan att behöva vänta på svar från mottagande enheten. Därefter är SPI också simpelt, fast inte lika simpelt eftersom mastern först måste använda chip select för att välja vilken enhet den vill prata med. I2C är simpel på det sättet att den är tydlig, men kräver också väldigt komplicerade protokoll som kan bli svåra att implementera för nybörjare.

5 DISKUSSIONER OCH SLUTSATSER

I denna sektion vägs resultaten mot varandra och en diskussion förs över vilka teknologier som passar systemet bäst. Utifrån denna diskussion dras en slutsats om vilken lösning som bör väljas för systemet.

5.1 Trådlös kommunikation

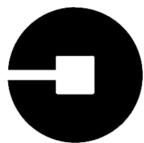
På grund av 802.11n-protokollets större räckvidd, högre dataöverförmingshastighet och bättre störtålighet lämpar sig den teknologin bättre för det system som ska utvecklas. Den stora fördelen med BLE är dess energieffektivitet, då bilen drivs med ett batteri skulle det hålla längre om BLE användes istället för 802.11n, dock är batteriets kapacitet inte en begränsning i detta fall. Ett fulladdat batteri räcker mer än väl för att utföra tillräckligt många köruppdrag före det behöver laddas om oavsett vilken trådlös kommunikationsteknologi som används.

5.2 Bussar

Eftersom beställaren i projektet ej satt krav på prestanda och budget på projektet anses de aspekterna komma sekundärt jämfört med simplicitet. Projektgrupperna är oerfarna inom kommunikaitionsområdet, och kommer inte ha så mycket tid att skriva kod för och felsöka kommunikationsmomentet i projektet. Med grunden i simplicitet anses UART vara den lämpligaste kommunikationsmodulen, eftersom den möter grundkraven som ställs på budget och prestanda för dataöverföring. Ett annat lämpligt alternativ hade kunnat vara SPI, eftersom det hade möjliggjort mer direktkommunikation mellan enheter, men eftersom det var mer avancerat än UART valdes det bort i detta fall.

6 REFERENSER

- [1] A. Holt and C.-Y. Huang, *802.11 Wireless Networks: Security and Analysis*. Springer, London, 2010, <https://link.springer.com/book/10.1007%2F978-1-84996-275-9#toc>.
- [2] A. Prasad and N. Prasad, *802.11 WLANs and IP networking / security, QoS and Mobility*. Artech House, Boston, 2005.
- [3] Raspberry Pi Foundation, “Raspberry Pi 3 Model B,” <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>, 2019, [Online; accessed February 21, 2019].
- [4] Broadcom Inc., “BCM4335,” <https://www.broadcom.com/products/wireless/wireless-lan-infrastructure/bcm4335#overview>, 2019, [Online; accessed February 21, 2019].
- [5] B. Benchoff, “Introducing the Raspberry Pi 3,” <https://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/>, 2016, [Online; accessed February 21, 2019].
- [6] M. Kathing, S. Bhattacharjee, and R. Rajkumari, “A Comparative study of Counter mode with Cipher block chaining Message authentication code Protocol (CCMP),” 2013, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.8575&rep=rep1&type=pdf>.
- [7] Bluetooth SIG, “Bluetooth Core Specification v4.0,” 2010, <https://www.bluetooth.com/specifications/archived-specifications>.
- [8] —, “Bluetooth Core Specification v5.1,” 2019, <https://www.bluetooth.com/specifications/bluetooth-core-specification>.
- [9] C. Gehrman, J. Persson, and B. Smeets, *Bluetooth Security*. Artech House, Boston, 2004, <https://ebookcentral.proquest.com/lib/linkoping-ebooks/detail.action?docID=227650>.
- [10] K. Townsend, C. Cufí, Akiba, and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. O'Reilly, 2014, https://books.google.se/books?id=24N7AwAAQBAJ&pg=PR2&dq=ISBN:9781491949511&hl=sv&cd=1&source=gbs_api#v=onepage&q=ISBN%3A9781491949511&f=false.
- [11] I. C. B.-S. . (https://creativecommons.org/licenses/by_sa/4.0)], “UART timing diagram,” https://commons.wikimedia.org/wiki/File:UART_timing_diagram.svg, 2016, [Online; accessed February 21, 2019].
- [12] B. Paradigm, “Introduction to I²C and SPI protocols,” <https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/>, 2018, [Online; accessed April 15, 2019].
- [13] N. Semiconductors, “UM10204 I²C-bus specification and user manual,” <https://www.nxp.com/docs/en/user-guide/UM10204.pdf/>, 2014, [Online; accessed March 27, 2019].
- [14] “The I²C Clock Speed – Why Accuracy Doesn’t Matter,” <https://www.i2c-bus.org/speed>, [Online; accessed April 15, 2019].
- [15] J. Lindblom, “Serial Communication,” <https://learn.sparkfun.com/tutorials/serial-communication/all>, [Online; accessed April 15, 2019].



Autonom taxibil

2021–07–09

UNTER

G FÖRSTUDIE STYRNING



UNTER

Reglerteori

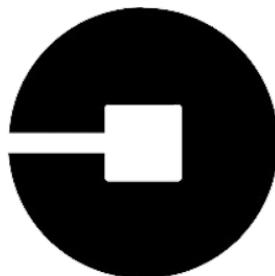
4 juni 2019

Förstudie

Olof Mlakar och Emil Mårtensson

4 juni 2019

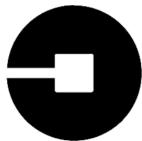
Version 1.2



UNTER

Status

Granskad	Olof Mlakar och Emil Mårtensson	2019-04-16
Godkänd		



UNTER

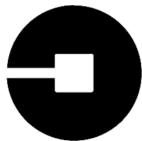
Projektidentitet

Grupp E-post: oloml269@student.liu.se

Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Johan Löfberg
Tfn: +46 13281304
E-post: johan.lofberg@liu.se

Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se



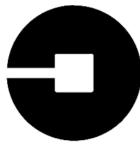
UNTER

Reglerteori

4 juni 2019

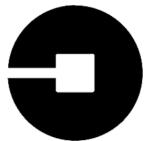
Sammanfattning

Denna förstudie är en del av kandidatprojektet i elektronik (TSEA56) på Linköpings Universitet. Syftet med studien är att simulera olika regulatorer för en autonom taxibil så att en optimal regleralgoritm kan användas senare i projektet. I denna studie så har programvaran Matlab använts tillsammans med simuleringsprogrammet Simulink för att testa de olika regulatorerna. De första försöken simulerades direkt i Matlab och gav goda resultat som verifierade att den teoretiska modellen *The Kinematic Bicycle Model* kunde användas för att simulera projektets fordon. Simuleringarna i Matlab var dock begränsade och därför inte tillräckliga för att ge ett entydigt svar på frågeställningen. För detta ändamål användes simuleringsprogrammet Simulink. Resultatet från simuleringarna i Simulink tyder på att en PD-regulator bör användas för att reglera gruppens fordon då den på ett snabbt och stabilt sätt kan följa en referenslinje.

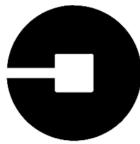


INNEHÅLL

1	Inledning	1
2	Frågeställning	1
3	Disposition	1
4	Källkritik	2
5	Beskrivning av bilen	2
6	PID-reglering	2
7	Modeller	3
7.1	Bicycle Model	3
7.2	Pure Pursuit	4
7.3	Stanley Method	4
8	Följa vägmarkering	4
8.1	Val av regulator	4
9	Simulering av algoritmer	5
9.1	Implementering av The Kinematic Bicycle Model	5
9.2	Simulering i Matlab	6
9.3	Begränsningar i Matlab	9
9.4	Simulering i Simulink	9
9.5	Begränsningar i Simulink	10
10	Resultat	12
10.1	P-reglering	12
10.2	PD-reglering	13
11	Diskussion/Slutsats	14
12	Referenser	15
13	Appendix	15
A	Kravspecifikation	15

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2019-03-04	Första utkast	O.M, E.M	O.M, E.M
1.0	2019-03-27	Första version	O.M, E.M	O.M, E.M
1.1	2019-04-10	Mindre rättelser	O.M, E.M	O.M, E.M
1.2	2019-04-16	Grammatiska kompletteringar	O.M, E.M	O.M, E.M



1 INLEDNING

Autonoma bilar har länge varit biltillverkarnas avlägsna dröm, men med tiden har teknologin blivit allt mer tillgänglig. Idag befinner sig bilarna i en testfas där olika omfattande experiment med autonoma bilar utförs i verkliga miljöer. Man har tidigare lyckats skapa autonoma bilar, de har dock medfört substansiella tekniska begränsningar. Den industriella kapplöpningen mellan biltillverkarna har gett upphov till diverse problem av olika natur. Vissa av problemen behöver lösas på politisk nivå innan den autonoma bilen kan bli en del av biltrafikens ekosystem.

En av de tekniska utmaningarna som en autonom bil ställs inför är att följa vägen. Bilen behöver utföra många krävande beräkningar, t.ex. linjedetektering (line detection) eller algoritmer som förutsäger andra aktörers rörelser. Eftersom alla dessa algoritmer endast har tillgång till en begränsad processorkraft så är det viktigt att varje algoritm är så kostnadseffektiv som möjligt. Av den anledningen är syftet med denna rapport att undersöka om man kan lösa några av den autonoma bilens reglertekniska problem med billiga reguleringsalgoritmer.

2 FRÅGESTÄLLNING

Denna förstudie är en del av kursen TSEA56 som går ut på att konstruera och programmera en autonom taxibil. Innan projektarbetet kan börja så ska tre förstudier genomföras av projektgruppen, en med inriktning på kommunikation, en med inriktning på sensorer och denna som är inriktad på reglerteknik. Denna studie syftar till att undersöka frågeställningen "Vilka övergripande styrmoder är nödvändiga för att roboten ska kunna utföra sitt uppdrag?"^[1]. Roboten är i detta fall den autonoma taxibilen.

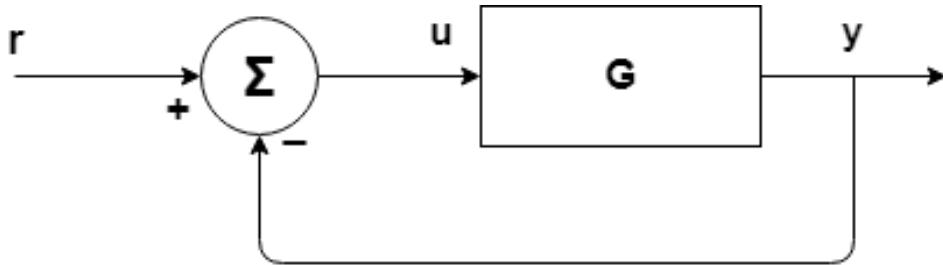
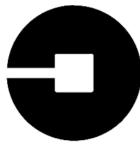
Följande frågor kommer att behandlas i studien:

- Vilken information krävs för reglering av bilen?
- Ska en modell användas och isåfall vilken?
- Vilken regulator är enklast att implementera samtidigt som den uppfyller kraven för stabilitet och snabbhet?

3 DISPOSITION

Här beskrivs studiens olika avsnitt.

I avsnitt 4 diskuteras trovärdigheten på de källor som har använts i studien. Avsnitt 5 avgränsar problemet och beskriver vilken data och hårdvara författarnas grupp förväntas kunna använda utan ytterligare bearbetning. Avsnitt 6-7 går igenom den grundläggande teorin som behövs för att lösa reglerproblemets och i avsnitt 8 tar rapporten upp den styrmod som anses vara mest relevanta för det valda projektet. Kapitel 9-10 beskriver den kvalitativa delen av studien där teorin från avsnitt 6-7 appliceras på frågeställningen och i avsnitt 11 presenteras rapportens slutsats tillsammans med en diskussion om det framtagna resultatet. I kapitel 6 och 8 utgår studien från ett återkopplat system som beskrivs i figur 1.



Figur 1: Ett återkopplat system.

4 KÄLLKRITIK

Denna studie är huvudsakligen skriven med hjälp av vetenskapliga artiklar och publicerade studier från bl.a. IEEE och LiU. Det medför att dess innehåll är högst tillförlitligt. Källorna är hämtade från Linköpings bibliotek och Google Scholar.

5 BESKRIVNING AV BILEN

Bilen som används i studien är fyrfjulsdriven och drivs av en elektrisk motor. En elektrisk servomotor bestämmer vinkelns på framhjulen och kan styrs med pulsbredden av en insignalpuls. Den elektriska drivmotorn styr vridmomentet som fördelar mellan bilens hjul och därmed bilens hastighet. Bilens position bestäms endast inomhus på platt underlag, hjulen antas därför ha kontakt med underlaget under alla tidpunkter. Bilens position bestäms med hjälp av en vidvinkelkamera för att bestämma vägen och bilens position samt en avståndssensor och en vinkelhastighetssensor på bakhjulet. Med hjälp av dessa sensorer så kan information som bilens position i förhållande till vägen, vägens mittpunkt, eventuella hinder på vägen och bilens hastighet beräknas och appliceras.

6 PID-REGLERING

Reglerteknik kan i dagligt tal beskrivas som ”Att få system att uppföra sig som man vill” och ett sätt att uppnå detta är genom en PID-reglering. En PID-reglering är en kombination av Proportionell (K_P), Integrande (K_I) och Deriverande (K_D) reglering [2]. Detta kapitel går igenom olika former av P, I och D-reglering.

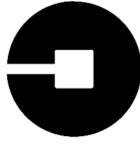
P-reglering innebär att man reglerar styrsignalen $u(t)$ med hjälp av reglerfelet $e(t)$, som beror på utsignalen $y(t)$ samt referenssignalen $r(t)$ och beskrivs av

$$e(t) = r(t) - y(t), \quad (1)$$

samt insignalnivån u_0 som beskrivs av

$$u(t) = u_0 + K_P e(t). \quad (2)$$

[2]



UNTER

Fördelarna med P-reglering är att den kan göra snabba regleringar, men den kan ha problem med att ta bort statinära fel. Ett sätt att eliminera det stationära reglerfelet är genom att introducera en integrerande del vilket i allmänhet leder till att utsignalen $y(t)$ går mot det önskade värdet. Detta kallas PI-reglering och beskrivs av

$$u(t) = u_0 + K_P e(t) + K_I \int_0^t e(\tau) d\tau. \quad (3)$$

[2]

Problemet med PI-reglering är att vid ökning av K_P och K_I genereras ett bättre resultat, men detta kan också leda till oscillativt beteende vilket gör systemet instabilt. För att motverka detta kan D-reglering introduceras i form av en PD-regulator enligt

$$u(t) = u_0 + K_P e(t) + K_D \frac{d}{dt} e(t) \quad (4)$$

eller en PID-regulator enligt

$$u(t) = u_0 + K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t). \quad (5)$$

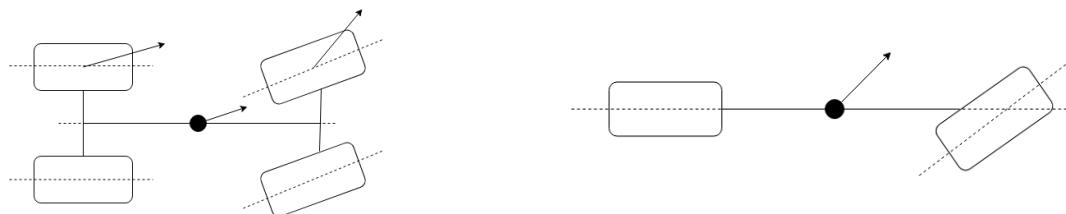
Genom att introducera D-reglering så kan man minska på oscillationerna och få ett stabilare system. [2].

7 MODELLER

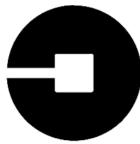
Detta kapitel går igenom hur reglerproblem kan lösas med hjälp av geometriska modeller som *The Kinematic Bicycle Model*, *Pure Pursuit* och *Stanley Method*.

7.1 Bicycle Model

En användbar modell inom fordonsreglering är cykelmodellen, eller på engelska *Bicycle model*. Det finns olika varianter på cykelmodellen, som den dynamiska (3 dimensioner) och den kinematiska cykelmodellen (2 dimensioner). I studien så kommer den kinematiska cykelmodellen, *The Kinematic Bicycle Model* att användas [3]. Poängen med att använda cykelmodellen är att ett fordon (bil) kan modelleras som en cykel med två frihetsgrader [4], beskrivs i figur 2.



Figur 2: Beskrivning av cykelmodellen



7.2 Pure Pursuit

Pure pursuit är en metod som styr ett rörligt fordon med hjälp av givna referenspunkter. Algoritmen beräknar vilken styrvinkel bilen bör ha för att följa en cirkelbana som korsar referenspunkten. Detta sätt att styra en bil på fungerar bra när bilen följer en relativt rak väg, men om bilen behöver göra snäva svängar uppstår ett oscillativt beteende [5].

7.3 Stanley Method

Stanleymodellen är en geometrisk styralgoritm som regleras genom en ickelinjärt återkopplad funktion. Den användes bl.a. av Stanford University i tävlingen "The DARPA Grand Challenge" [6].

8 FÖLJA VÄGMARKERING

Den största reglerutmaningen gällande den autonoma taxibilen kommer vara att få den att följa vägen på ett säkert och stabilt sätt. Detta inkluderar körning i kurvor och rondeller. Bilen ska helt autonomt kunna identifiera vägens mittpunkt samt reglera styrningen kring den.

Problemet att få en bil att följa en väg kan beskrivas på följande sätt:

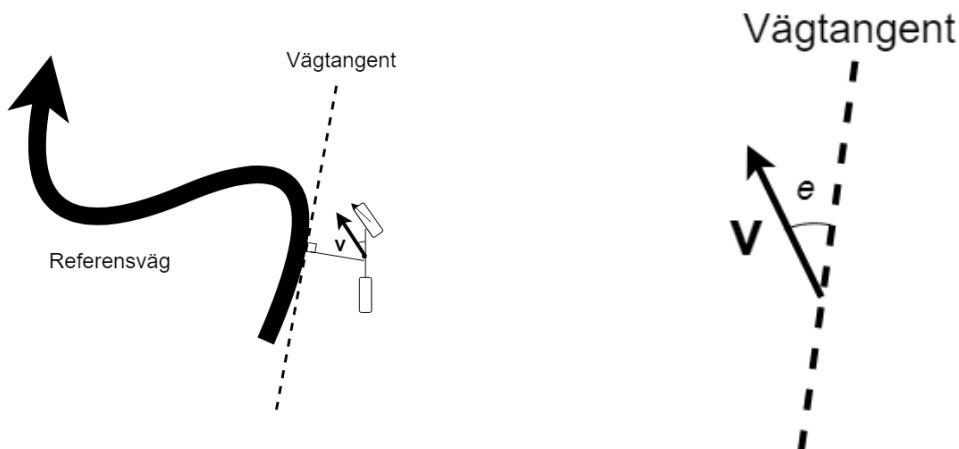
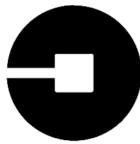
En fyrhjulsdriven bil med ett styr servo ska följa en bilväg genom att placera sig mitt i högerfilen. Vägens mitt- och sidlinje är approximativt två diskreta vektorer med ändlig längd. Ta fram en lämplig regleringsalgoritm som tillfredsställer dessa specifikationer.

8.1 Val av regulator

För att att erhålla en effektiv styrning måste regleralgoritmen uppfylla olika krav. Det finns tre viktiga egenskaper för reglering av fordon i trafik. Det reglerade systemet måste vara stabilt, den får inte ha några trafikfarliga överslängar (d.v.s. bilen måste hålla sig inom sitt körfält) och systemet måste också vara snabbt nog för att klara av att hantera skarpa svängar. Högre hasigheter ställer också högre krav på regleringsalgoritmen. Målet är alltså att skapa ett snabbt, stabilt system som inte har trafikfarliga "överslängar". Bilen bör alltså varken köra över i mötande fil eller köra av vägen.

En P-regulator kan användas för att uppnå önskad snabbhet, men det finns en risk att systemet då får för stora överslängar (vilka kan minskas med att lägga till en deriverande del). Det finns också en risk att bilens position driver åt något håll, vilket kan motverkas genom att lägga till en integrerande del i regulatorn (se avsnitt 6).

Med den kinematiska cykelmodellen kan bilens hastighet och riktning beräknas, vilket i sin tur kan användas för att simulera den reglerade bilens beteende. Det finns två reglerfel som är intressanta att använda i en regulator. Den ena är vinkelskillnaden mellan bilens hastighetsvektor och en tangent till den identifierade vägmarkeringen (figur 3). Problemet med detta är att PD-regulatorn får ett stationärt fel eftersom bilen nödvändigtvis inte behöver vara på körbanan för att vinkelskillnaden ska bli noll. En PID-regulator skulle alltså inte kunna kompensera för det stationära felet. Det andra reglerfelet är den laterala skillnaden mellan bilens position och vägmarkeringen. En PD-kontroller som använder detta reglerfelet slipper det ovannämnda stationära felet, men tar inte hänsyn till framtida laterala fel som skulle orsakas av ett vinkelfel, vilket kan öka bilens oscillativa beteende. Utifrån detta resonemang är det fördelaktigt att ta fram ett system som tar hänsyn till både den laterala skillnaden och vinkelskillnaden.



Figur 3: Vinkelskillnaden e mellan tangenten och bilens riktning

Nedan defineras studiens tänkta variabler tillsammans med dess betydelse:

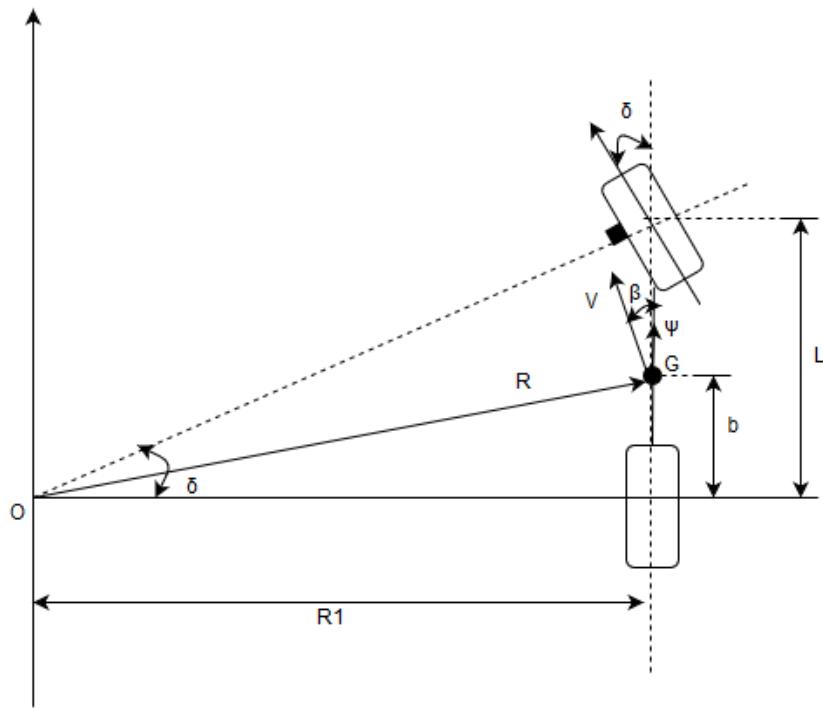
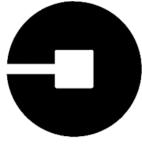
- $u(t)$ (Styrsignal): Beskriver insignalpulsen till bilens styr servo, ett högre värde ger en större utfallsvinkel på servot. (För enkelhetens skull antas styrservot göra momentana vinkeländringar)
- $r(t)$ (Referenssignal): Den önskade laterala positionen på vägen som bilen ska följa.
- $v(t)$ (Störsignal): Brus i systemet som kan påverka dess egenskaper.
- $y(t)$ (Utsignal): Bilens laterala position på vägen.

9 SIMULERING AV ALGORITMER

För att undersöka kvalitativa egenskaper hos de presenterade regleringsalgoritmerna så används simuleringsverktyg i programmet Matlab. Följande avsnitt går igenom metoden och implementeringen av cykelmodellen samt implementeringen i Matlab. Då regulatorerna simuleras i Matlab så förekommer inget stationärt fel att påverka regulatorerna. Därför simuleras inte en PID-regulator då resultat blir samma som för en PD-regulator.

9.1 Implementering av The Kinematic Bicycle Model

Regleralgoritmen har konstruerats utifrån The Kinematic Bicycle Model, figur 4 beskriver de variabler som används. R1 är avståndet från referenslinjen till bilens mittpunkt G (i meter m), L är bilens längd (m) mätt mellan hjulens mittpunkter och b är avståndet (m) från bilens mittpunkt till bakhjulets mittpunkt. Vinkel delta (δ) är hjulets styrvinkel, vinkel beta (β) är bilens styrvinkel och vinkel ψ är bilens lutningsvinkel. V är bilens hastighets vektor [7].



Figur 4: Cykelmodellen.

I Matlab simuleringarna, som beskrivs i detalj i avsnitt 9.4, användes följande ekvationer vid beräkning av positionerna X och Y:

$$\beta = \frac{\delta * b}{L} \quad (6)$$

$$\dot{\psi} = \frac{V}{b} * \sin(\beta) \quad (7)$$

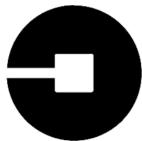
$$\dot{X} = V \cos(\psi + \beta) \quad (8)$$

$$\dot{Y} = V \sin(\psi + \beta). \quad (9)$$

Formlerna är framtagna från cykelmodellen i figur 4 [3]. Formlerna ger \dot{X} , \dot{Y} och $\dot{\psi}$ vilket är derivatan med avseende på tiden för positionerna X, Y och vinkeln ψ . För att ta fram de sökta variablerna integreras de.

9.2 Simulering i Matlab

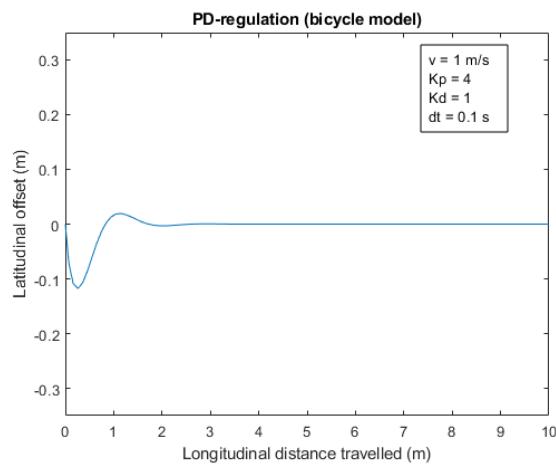
De framtagna rörelseekvationerna simulerades direkt i Matlab. Matlab användes för att det kan hantera avancerade matematiska funktioner samt rita upp grafer som gör det enkelt att förstå resultatet.



UNTER

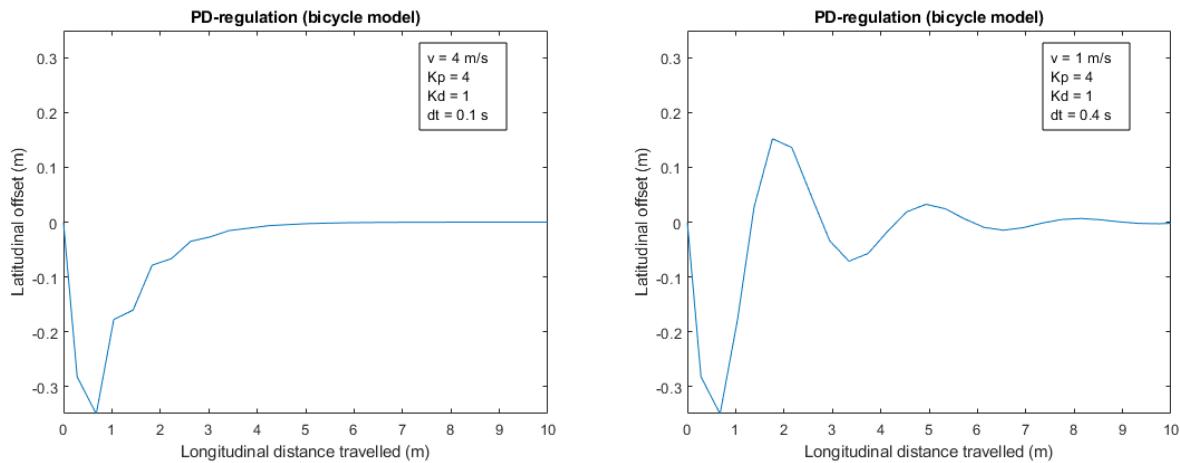
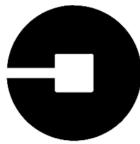
Simuleringen i Matlab gjordes för att testa vilket resultat man kunde förvänta sig från de olika regulatorerna. Det gjordes också för att kontrollera att modellen kunde appliceras på frågeställningen innan den mer avancerade simuleringen i Simulink påbörjades.

Simuleringarna av PD-regleringen visar att en PD-reglering med bilens styrvinkel som styrsignal ger en stabil reglering utan ett stationärt reglerfel. Figur 5 nedan visar ett enkelt fall där referenslinjen är x-axeln och bilen börjar med en vinkelavvikelse $\psi = -45^\circ$ och lyckas stabilisera sig efter en kort tid. Simuleringen i figur 5 är högst approximativ och därmed inte lämplig för att optimera regulatorn.



Figur 5: Tidigt test av PD-reglerad cykelmodell

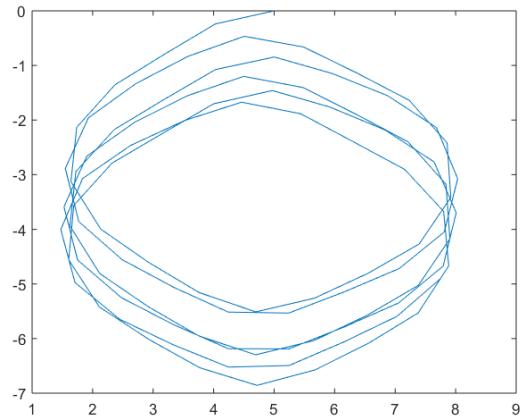
Cykelmodellen är begränsad till en konstant hastighet, i figur 5 undersöktes hastigheten 1 m/s. Figur 6 visar simuleringen av bilen vid en högre hastighet respektive samplingstid.



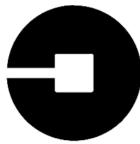
Figur 6: PD-Reglering med hastighet 4 m/s (vänster) respektive samplings tiden 0.4 s (höger)

En högre hastighet leder alltså till mer tvära regleringar, eftersom högre hastigheter får felet att växa snabbare. En längre samplingstid ger också ett ryckigt beteende, men jämfört med den högre hastigheten så är derivatan på felet mycket lägre. Den högre sampletiden gör istället att regulatorn inte reagerar lika ofta och utför en längsammare reglering. För enkelhetens skull användes sampletiden för att approximera derivatan också, något som simulink separerar på naturligt.

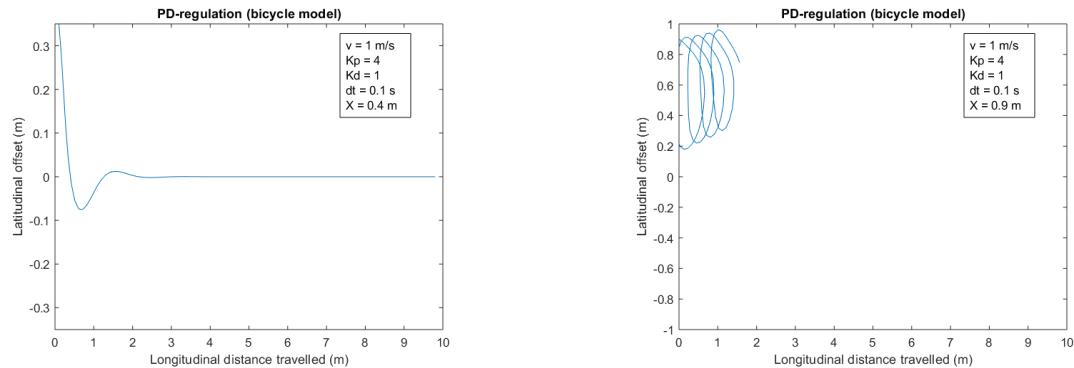
Några andra begränsningar med denna modell är bl.a. hjulets styrutslag. I verkligheten så kan framhjulen på en bil sällan svänga mer än några tiotal grader medan modellen som används kan ha ett styruslag på 360° . För att begränsa detta implementerades en If-sats i koden för att kunna reglera styrutslaget till maximalt 90° . Detta löste en del av problemet men det ledde också till att styrutslaget fastnade på 90° och fordonet började köra runt i en cirkel vilket beskrivs i figur 7.



Figur 7: Styrutslag fast på 90°



Slutligen så begränsas denna modell av avståndet till startpositionen. Vid stora avstånd från referenslinjen försöker fordonet snabbt svänga in sig mot referenslinjen genom att vinkla hjulet mot referenslinjen. Detta leder till begränsad översläng med PD reglering och stora oscillationer vid användning av P-reglering. Vid stora avstånd från referenslinjen så försöker bilen vinkla hjulen 90° mot referenslinjen vilket medför att den fastnar i en cirkel, vilket togs upp i föregående stycke. Resultatet kan ses i figur 8.



Figur 8: PD-Reglering vid olika avstånd från referenslinjen.

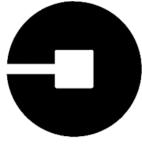
9.3 Begränsningar i Matlab

Pågrund av nämnda begränsningar är denna simulering endast lämplig för en kvalitativ analys av regleringen. För att skapa en simulering som är mer verklighetstrogen bör denna modell återskapas i simulink, som kan utföra kraftigare derivataberäkningar samt variera samplingstiden dt på ett smidigt sätt. Begränsningen av hjulets styrutslag bör också ändras till ett mer realistiskt värde i senare simuleringar.

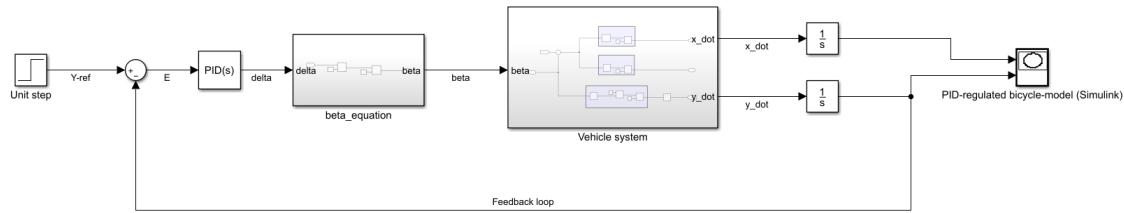
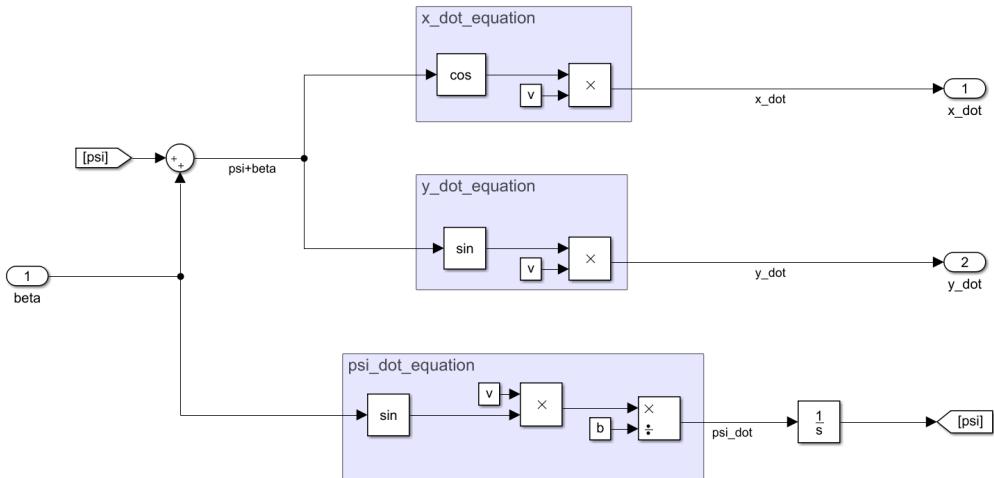
9.4 Simulering i Simulink

Simuleringen av rörelseekvationerna i Matlab gav bra resultat och visade att modellen kunde användas för att testa olika regulatorer. Den räcker dock inte för att ge ett absolut svar på frågeställningen "Vilken reglering är bäst för att bilen ska följa vägen på ett säkert och stabilt sätt". För att svara på det så måste ett bättre simulatingsverktyg användas med en noggrannare approximation av derivatan. I studien så användes Simulink för detta ändamål. Simulink är ett simuleringsprogram i Matlab som gör det enkelt att implementera och testa regulatorer i ett grafiskt användargränssnitt.

I figur 9 visas hela modellen i sin utvecklingsmiljö och i figur 10 respektive 11 visas de två delsystemen i detalj.

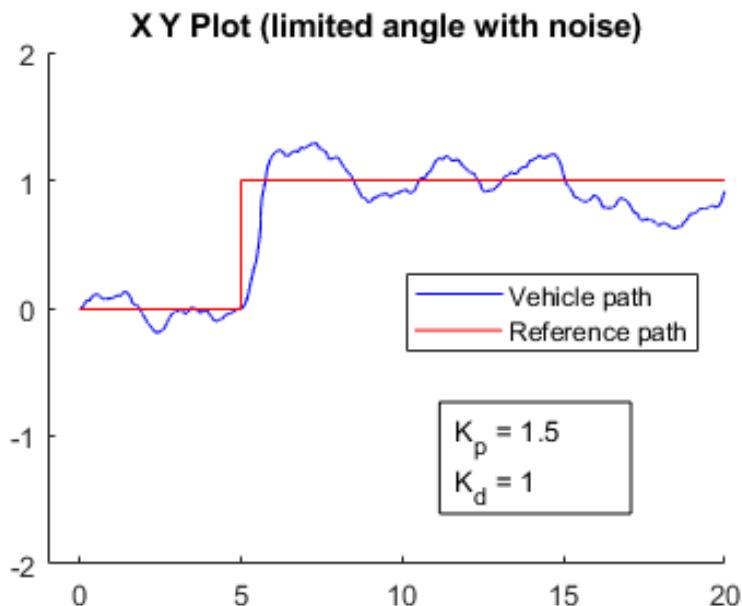
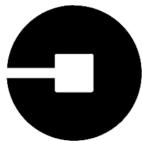


UNTER

**Figur 9:** Simulink modellen i dess utvecklingsmiljö**Figur 10:** Systemet "Beta equation" i detalj**Figur 11:** Systemet "Vehicle system" i detalj

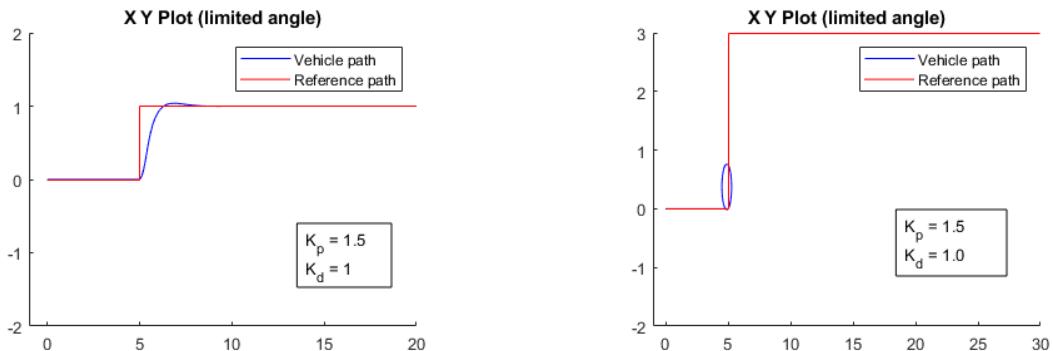
9.5 Begränsningar i Simulink

Några begränsningar med simuleringarna i Simulink är som nämntes i inledningen att modellen inte tar hänsyn till stationära fel eller mätbrus. Resultatet vid introduktion av mätbrus kan ses i figur 12.

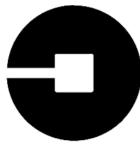


Figur 12: Det begränsade systemet med mätbrus

Avståndsbegränsningen från referenslinjen gjorde sig även synlig i simuleringar med Simulink, trots att ett mer realistiskt maxvärde på hjulets styrutslag (max styrutslag på hjulet var 45°) implementerades. Detta antyder att regleralgoritmen i sig är begränsad till ett maximalt avstånd från referenslinjen. Resultatet kan ses i figur 13 där avståndet från referenslinjen är tre gånger så stort (vid start) som normalt. Resultatet blir att fordonet fastnar i en loop som liknar det resultatet som uppmättes i Matlab simuleringarna (se figur 7).



Figur 13: Reglering då fordonet placeras 1 m från referenslinjen (vänster) och reglering då fordonet placeras 3 m från referenslinjen (höger).

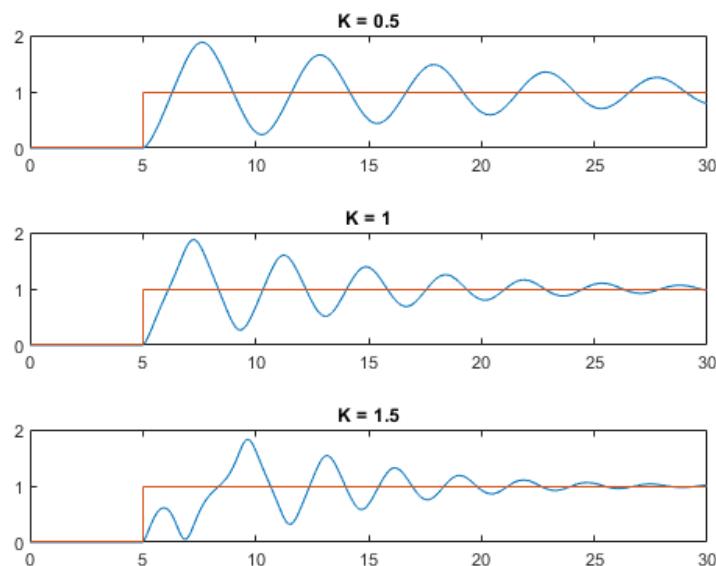


10 RESULTAT

I detta avsnitt redovisas systemets beteende med olika parametrar och referensvägar.

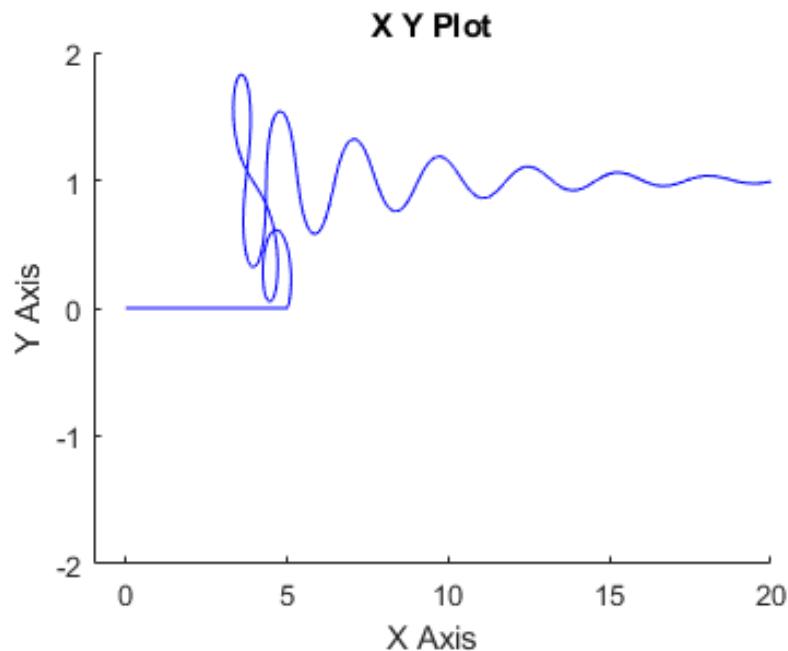
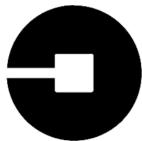
10.1 P-reglering

P-reglering i Simulink modellen testades ett flertal gånger och visade att ett högre värde på K_p gav ett oscillativt beteende. Detta beskrivs i figur 14 där den blå linjen är fordonets reglering och den röda linjen är referenslinjen som fordonet ska följa.



Figur 14: P-regulator med tre olika K .

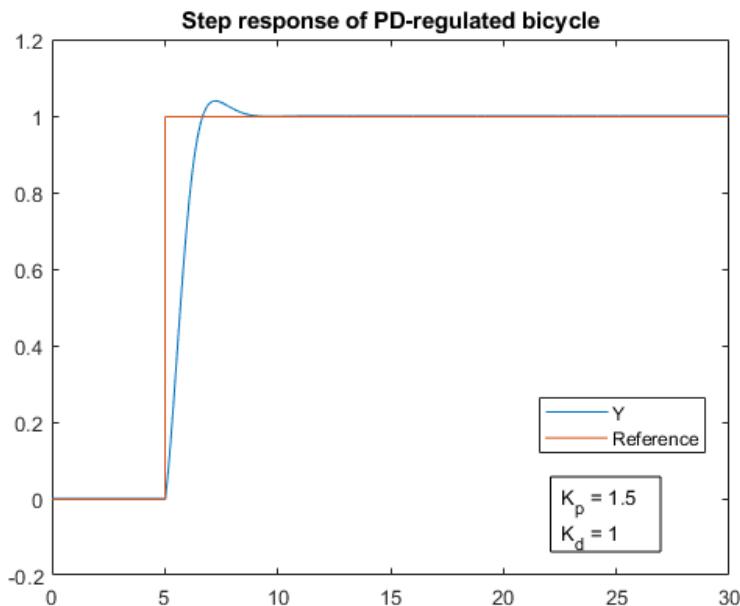
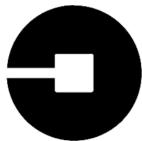
Vid ett lågt K ($K = 0.5$) svänger fordonet in sig långsamt men ändå mot referenslinjen. Om K ökas till 1.0 så oscillerar fortfarande fordonet men det svänger in sig snabbare på referenslinjen vilket kan ge intrycket av att användning av ett tillräckligt stort K kan ge en tillräckligt stabil reglering med en P-regulator. Den sista grafen i figur 14 visar dock att fordonet börjar bete sig underligt vid $K = 1.5$. Figur 15 visar att fordonet börjar köra runt på ett minst sagt instabilt sätt.



Figur 15: Fordonet körbana vid $K = 1.5$.

10.2 PD-reglering

PD-reglering i Simulink modellen testades ett flertal gånger och visade upprepade gånger att den snabbt ställer in sig efter referenslinjen. I figur 16 visas en av testerna som gjordes av en PD-regulator med värdena $K_p = 1.5$ och $K_d = 1.0$ (den röda linjen är referenslinjen och den blå är fordonets reglering). Det syns tydligt att den snabbt och stabilt ställer in sig efter referenslinjen med endast en liten översläng som belastning. Viktigt att tänka på vid denna simulering är att en PD-regulator är känslig för mätbrus.



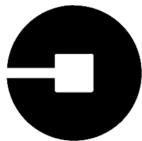
Figur 16: PD-regulator.

11 DISKUSSION/SLUTSATS

Resultaten i avsnitt 10 tyder på att en PD-regulator är den minst komplexa regleralgoritmen som kan användas och ändå uppnå en godtagbar stabilitet. Denna slutsats kan dras utifrån graferna i figur 14, 15 och 16 som visar att PD-regulatoren ställer in sig efter referenslinjen på en acceptabel tid utan oscillationer. Notera dock att PID-regulatoren skulle gett ett lika bra resultat men då varken stationära fel eller mätbrus uppkommer vid simuleringarna så finns det ingen direkt anledning att implementera den. Detta är dock en simulering av modellen och stationära fel samt mätbrus kan uppkomma vid testning av den autonoma taxibil där denna regulator ska implementeras. Om detta sker så kan det vara nödvändigt att implementera en PID-regulator istället för en PD-regulator. Om det skulle uppstå högfrekventbrus vid testning så kan det även vara nödvändigt att implementera ett lågpassfilter.

Utifrån resultaten i avsnitt 8, 9 och 10 kan man dra följande slutsatser:

- För att skapa en regleralgoritm behövs information om fordonets storlek, längd, position och hastighet samt styrvinklar för fordonets kropp, hjul och lutningsvinkel.
- En modell bör användas och den som passar studien bäst är den så kallade cykelmodellen.
- Den regulator som enklast kan implementeras samtidigt som kraven på stabilitet och snabbhet uppfylls är en PD-regulator.

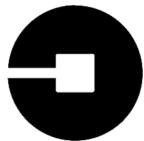


12 REFERENSER

- [1] M. Krysander, “Kandidatprojektet skrivuppgift,” <http://www.isy.liu.se/edu/kurs/TSEA56/forelasning/Skrivuppgift.pdf>, [Online; accessed February 19, 2019].
- [2] T. Glad and L. Ljung, *Reglerteknik Grundläggande Teori*, 4th ed. Lund: Studentlitteratur, 2006.
- [3] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, “Kinematic and dynamic vehicle models for autonomous driving control design,” *IEEE Intelligent Vehicles Symposium*, 2015.
- [4] E. Ryding and E. Öhlund, “Lane keeping aid - ett förarstödjande system för bilar,” *LITH-ISY-EX-3207-2002*, 2002.
- [5] T. S. W. W. J. Wang, T. M. Hsu, “The improved pure pursuit algorithm for autonomous driving advanced system,” *IEEE IWCIA*, 2017.
- [6] J. M. Snider, “Automatic steering methods for autonomous automobile path tracking,” *CMU-RI-TR-09-08*, 2009.
- [7] G. Genta, *Motor Vehicle Dynamics, Modeling and Simulation*. 5 Toh Tuck Link, Singapore 596224: World Scientific Publishing Co. Pre. Ltd., 2006, vol. 43.

13 APPENDIX

A KRAVSPECIFIKATION



Autonom taxibil

2021–07–09

UNTER

H FÖRSTUDIE SENSOR



UNTER

Förstudie

4 juni 2019

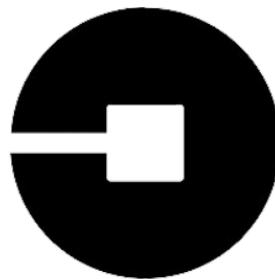
Förstudie

Vägnätsnavigering med hjälp av bilddata

Jonathan Carlin och Dennis Österdahl

4 juni 2019

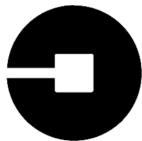
Version 1.0



UNTER

Status

Granskad	JC, DÖ	4 juni 2019
Godkänd		

**UNTER****Projektidentitet**Grupp E-post: jonca673@student.liu.se

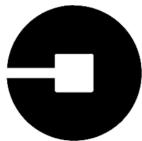
Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Karl Holmquist
Tfn:
E-post: karl.holmquist@liu.se

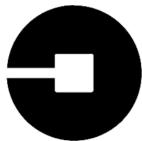
Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

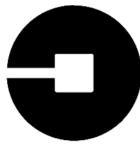
Namn	E-post
Jonathan Carlin	jonca673@student.liu.se
Dennis Österdahl	denos835@student.liu.se

**UNTER****INNEHÅLL**

1	Inledning	1
1.1	Bakgrund	1
1.2	Syfte	1
1.3	Frågeställningar	1
1.4	Avgrensningar	1
1.5	Metod	2
1.6	Struktur	2
2	Kunskapsbas	2
2.1	Vad är datorseende?	2
2.2	Hur fungerar datorseende	3
2.3	Navigeringen i ett vägnät med hjälp av datorseende	3
2.3.1	Maskning	3
2.3.2	Gråskalning	4
2.3.3	Gaussiskt filtrering	4
2.3.4	Kantdetektion	5
2.3.5	Hough-Transform	5
2.4	Programpaketet OpenCV	6
3	Resultat	7
3.1	Bildupplösning	7
3.2	Bildbehandlingstid	7
3.3	Bildberäkningar	9
4	Slutsats	10
5	Referenser	11
6	APENDIX	12

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2019-03-04	Första utkast	Grupp 9	JC, DÖ
0.2	2019-03-27	Andra utkast	Grupp 9	JC, DÖ
1.0	2019-04-17	Slutversion	Grupp 9	JC, DÖ



1 INLEDNING

I en allt mer digitaliseringad värld har datorn fått en allt större betydelse inom den teknologiska utvecklingen. Inom många olika industrier är datorn till stor hjälp, alltifrån flygindustrin till daglig detaljhandel använder sig av datorer som hjälpmittel. Att med hjälp av en dator samla in och tolka bilder har länge setts som en viktig teknik för den framtida tekniken. Med datorns hjälp att behandla bilder kan stora framsteg inom många industrier som exempelvis fordonsindustrin göras. Då en dator har en snabbare responstid än vad en människa har skulle det kunna vara möjligt att med hjälp av datorseendet förhindra diverse trafikolyckor. Exempelvis så hinner en dator reagera och få bilen att bromsa i fall något oväntat skulle inträffa framför fordonet.

Denna förstudie beskriver datorns förmåga att ta in en bild, behandla den för att sedan ge en tolkning som hjälper användaren. Förstudien är uppdelade i en beskrivande del där konceptet datorseende förklaras och vilka användbara bildbehandlingsoperationer som används inom området. Resultatdelen av förstudien redogör hur navigering i ett vägnät kan ske med hjälp av datorseende.

1.1 Bakgrund

Förstudien är tänkt att agera som ett underlag för utförandet av kandidatprojekt, TSEA56, vid Linköpings Universitet där en autonom taxibil ska konstrueras.

1.2 Syfte

Syftet med förstudien är att undersöka hur en kamera och datorseende kan användas för att navigera i ett vägnät. Vad kommer krävas för att det ska vara möjligt och vad det finns för begränsningar med tekniken?

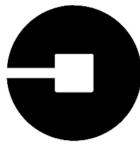
1.3 Frågeställningar

Förstudien ämnar att besvara följande frågor:

- Vad är datorseende och hur fungerar det?
- Hur kan datorseende användas för navigering i ett vägnät? Hur många bilder behöver behandlas och hur stor upplösning behövs?
- Finns det något färdigt programpaket för datorseende?

1.4 Avgränsningar

Förstudien begränsar sig till att fokusera på färdiga programpaket för datorseende, hur dessa paket kan användas inom tekniken och inte hur nya programpaket kan utvecklas. Behandlingen av datorseende kan ske olika fort beroende på tillgänglig beräkningskraft hos inblandade komponenter. Den tillgängliga beräkningskraften antas vara den hos en Raspberry PI3 och dess förmåga att behandla bilddata.



1.5 Metod

För att besvara de uppställda frågeställningarna används vetenskapliga tidskriftartiklar inom området tillgängliga via Linköpings Universitetsbibliotek. Genom att läsa relevanta artiklar inhämtas information om vad datorseende är och de olika metoder för datorseende som existerar idag. På ISY:s databladsserver Vanheden finns datablad för kameror som kan användas för att ta de bilder som datorseendet behöver.

1.6 Struktur

Förstudien behandlar inledningsvis en allmän beskrivning om datorseende, hur datorseende fungerar och vad datorseende innebär. Därefter följer en del där implementationen av datorseende sker med hjälp av färdiga programpaket. Slutligen behandlar rapporten förmågan att behandla bilddata på en Raspberry PI3.

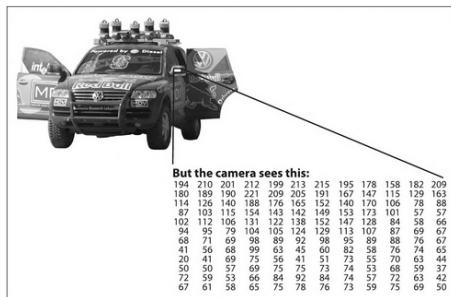
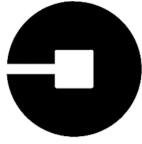
2 KUNSKAPSBAS

För att kunna avgöra vilken typ av datorseende som är mest lämpad för navigering i vägnät med hjälp av kamera krävs viss kunskap inom datorseende och vilka olika programpaket som finns. Genom en övergripande beskrivning över hur datorseende fungerar och vad det finns för tillgängliga programpaket undersöks de möjligheter som finns och hur lämpliga de är för vägnavigation.

2.1 Vad är datorseende?

Datorseende innebär att bild eller video som är insamlad av en kamera kan tolkas av ett datorprogram på ett sådant sätt att datorprogrammet kan dra en slutsats om vad som visas i bilden. Ur ett mänskligt perspektiv kan man lätt peka ut vad som finns i bilden. Om bilden är på en väggkorsning är det ofta lätt att på bilden peka ut var bilar och fotgängare är, om trafikljuset är grönt eller rött eller om fotgängaren är man eller kvinna. Människan kan kontextuellt tolka en bild med tidigare erfarenheter för att veta vad som söks i bilden. Människan kan även finna objekt i kontext, exempel en bil på en bild bör vara på marken på en väg och inte i luften [1].

En dator däremot har det inte lika lätt. Datorn ser endast bilden som en stor serie av nummer så som i figur 1 och kan utifrån dessa nummer inte dra någon slutsats kring vad bilden förmedlar. Datorseende är tekniken att med hjälp av algoritmer sammanställa dessa nummer till den situation bilden beskriver [1].



Figur 1: Vad en människa ser och vad en dator ser [1]

Enligt Richard Szeliski tror människor ofta att datorseende är lättare än det faktiskt är. Vi människor är så vana att konstant få in synintryck att man lätt glömmer hur komplext vårt synsinne är [2]. Datorseende kan efterliknas att beskriva en bild för en blind person, datorn får informationen om bilden digitalt medan man till en blind person muntligt skulle försöka beskriva vad bilden föreställer [1].

2.2 Hur fungerar datorseende

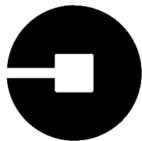
Datorseende fungerar som nämnt ovan att en bild för en dator endast är en stor mängd nummer. Dessa nummer måste på något sätt behandlas för att datorn ska kunna dra slutsatser om vad bilden representerar. Det visar sig vara svårt för datorseendealgoritmer att ta fram all relevant data ur en bild. Begränsningar införs lämpligen för att specificera vad i bilden som är av intresse att detektera. Fler begränsningar medföljer färre problem algoritmen måste hantera. Färre problem medföljer att algoritmen blir mer exakt och resultatet av algoritmen och datorseendet då mer tillförlitligt.[1] Implementationen av datorseendealgoritmer stöter dock på flera problem som måste lösas. Datorseendealgoritmer måste handskas med olika typer av störningar i bildens kvalité. Bilden kan vara suddiga om kameran eller det objekt som bilden togs på rör sig. Ljusreflektioner och förvidningar i kameralinsen kan göra bilden missvisande. För att motverka dessa störningar baserar sig de flesta datorseende algoritmer på statistisk analys, även om en del av bilden är missvisande så är bilden i sin helhet mestadels korrekt.[2]

2.3 Navigeringen i ett vägnät med hjälp av datorseende

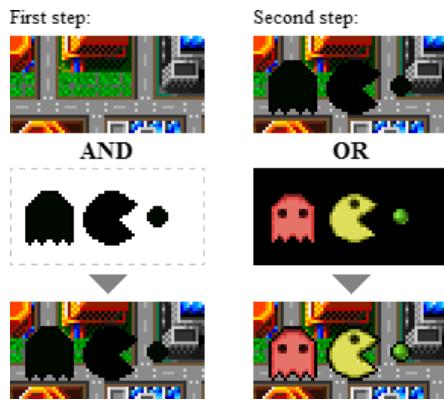
Med hjälp av etablerade verktyg kan navigation i ett vägnät genomföras. Datorseende använder sig av flera olika bildbehandlingsoperationer för att göra en bild lämplig att applicera datorseende på. I nedanstående rubriker presenteras vilka bildbehandlingsoperationer som kan användas vid navigering i ett vägnät. Processen från att en bild togs, de operationer som kan appliceras på bilden för detektion av väglinjer till att reglera på dessa linjer.

2.3.1 Maskning

Med maskning så beräknar man om varje pixelvärde i en bild enligt en maskningsmatris med hjälp av bitvis operation. Masken kan antingen vara lika stor som bilden och inkludera samt exkludera områden som enligt bild 2 eller vara av mindre storlek som i bild 4 och iterera över bildens pixlar med viktade värden.[3]. En vanlig maskning inom datorseende är binär maskning, se figur 2. De pixelvärden som i den riktiga bilden ej är av intresse representeras i en bitkarta med pixelvärde satt till 0. De pixlarna av intresse erhåller pixelvärde 1. Ytterligare en maskning sker där de



tidigare pixelvärdena satta till 0 sätts till 1 och de värdena satta till 1 sätts till 0. Således erhålls två bildtyper. En där enbart bakgrunden representeras och en där intresseområdet representeras[4].



Figur 2: Exempel på binär maskning

2.3.2 Gråskalning

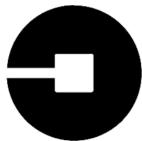
Gråskalning innebär att man konverterar om en bild från RGB färgskala till en enfärgsskala. Varje pixel erhåller ett värde mellan [0, 255] där 0 representeras av färgen svart och 255 färgen vit. Bildens pixelvärdet lagras sedan i en endimensionell matris. RGB-färgskalan representeras av en multidimensionell matris, så kallad tensor, medan gråskalan enbart består av en endimensionell matris[5]. För vidare datorseendetillämpningar t.ex Hough-transform är det användbart att ändra färgskalan till en enfärgsskala då detta underlättar för beräkningar[6].



Figur 3: Vanlig och gråskalad bild

2.3.3 Gaussiskt filtrering

Vid specifik detektering av ett objekt i en bild är det viktigt att kunna urskilja vad som är av intresse att detekteras och vad som kan försummas. Detta genomförs med hjälp av kantdetektion, se sektion 2.3.4. För att kantdetektionen ska kunna genomföras på ett så bra sätt som möjligt är det viktigt att ta bort onödigt brus i bilden. Detta då beräkning av gradienten framhäver höga frekvenser och därav förstärker bruset. Det är då av stor vikt att innan utförandet av kantdetektion genomföra en lågpassfiltrering för att filtrera bort de högfrekventa bruskällorna.



Gaussiskt brusfiltrering är det enda separerbart cirkulära symmetriska filtret, vilket betyder att det är det enda filtret som kan utföras först i x-led och sedan i y-led samtidigt som det är symmetrisk i cirkulär mening. Att kunna utföra filtreringen först i x-led och sedan i y-led medför en tidskomplexitet av $2N$ jämfört med andra filter som har en tidskomplexitet på N^2 , gaussisk filtrering är alltså snabb sett till växande filterstorlek.[2]

Gaussisk filtrering i två dimensioner [1] använder en sammansättning av två gaussiska funktioner

$$G(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}. \quad (1)$$

Filtreringen utförs med en mask av en vald storlek där varje index tilldelar ett viktat värde enligt normalfördelning, se figur 4, till pixlar i bilden. Den pixel som för tillfället behandlas är i mitten av masken och tilldelas efter filtrering ett nytt värde som är ett viktat medelvärde av de pixlar masken verkade över. Detta medför att närliggande pixlars värden rör sig mot varandra, konstrasten blir då mindre och bilden därmed suddigare.[2]

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Figur 4: 5x5 mask med normalfördelning.[3]

2.3.4 Kantdetektion

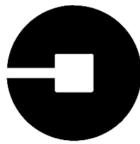
Målet med kantdetektion är att identifiera en plötslig förändring i en bild. Kanterna hjälper att extrahera information för att känna igen objekt och återskapa geometriska former. Kanterna detekteras genom diskontinuitet i ytnormalen, djupet, ytfärgen och ytans belysning [5]. Således kan en kant definieras som en position med snabb intensitetvariation, exempel den branta lutningen som uppkommer från ett plant fält till ett berg. Matematiskt kan lutningen och variationen på ytan beskrivas genom dess gradient [2]

$$J(x) = \nabla I(x) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) \quad (2)$$

där vektorn J pekar i riktningen mot den brantaste lutningen. Magnituden indikerar på lutningen eller styrkan i variatonen medan riktingen är vinkelrätt mot den lokala konturen.[2] En förfinad variant på kantdetektioner kallas för Canny edge detectors. Första derivatan beräknas i x och y och kombineras sedan till fyra riktningsderivator. De punkterna vars riktninginsderivator håller det lokala maximat blir sedan kandidater till skapandet av kanter. Canny-algoritmen försöker såledse skapa individuella kantkandidatpixlar till konturer. Dessa konturer formas genom att lägga till ett "hysteresis threshold" till pixlarna. Detta innebär att det finns två tröskelvärden, ett högre och ett lägre. Om en pixel har större gradient än den högre tröskeln accepteras den som en kantpixel. Om pixeln har lägre gradient än det undre tröskelvärdet förkastas den pixeln som kandidat. Om en pixelgradient ligger mellan de båda trösklarna accepteras den enbart om den är sammankopplad med en pixel som har en gradient högre än det övre tröskelvärdet [1].

2.3.5 Hough-Transform

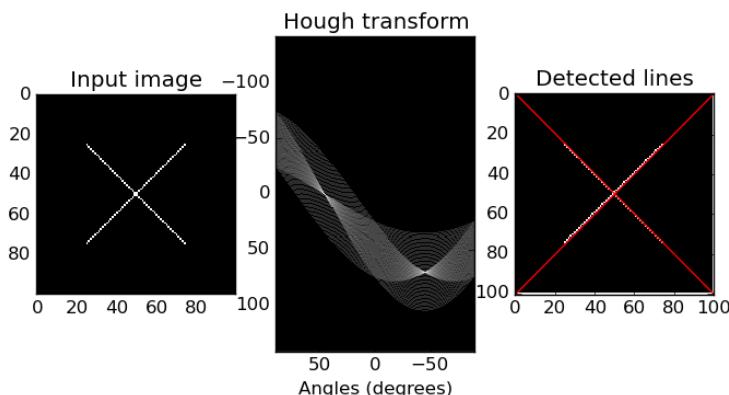
Hough transformen är en robust transform för att detektera strukturer i bilder. Transformen fungerar bäst på att detektera rät linjära strukturer men kan detektera alla strukturer som har en känd parameterform.[5] Transformen tillämpas



lämpligen på en bild som genomgått kantdetektion och är binär, se längst till vänster i figur 5. Transformen utgår ifrån att en pixel i bilden kan ses som en koordinat (x, y) och att en pixel som inte är 0 kan ingå i en mängd möjliga linjer. Pixeln parametreras med polära koordinater 3 och genom att variera värdena för ρ och θ kan alla möjliga linjer som innehåller pixeln på koordinaten (x, y) bestämmas [1].

$$\rho = x \cos \theta + y \sin \theta. \quad (3)$$

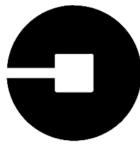
Varje möjlig linje representeras av en punkt i (ρ, θ) koordinatsystemet och alla möjliga linjer tillsammans bildar en sinusvåg i $\rho\theta$ -planet. Om ovanstående procedur utförs för alla icke nollställda pixlar i bilden erhålls en mängd vågor i $\rho\theta$ -planet som enligt mitten av figur 5. De punkter där flest vågor skär varandra är det störst sannolikhet att en linje i ursprungsbilden finns. (ρ, θ) -koordinaterna i dessa punkter utgör då värdena för den parametriserade linjen i (x, y) -planet. På så sätt kan linjerna i ursprungsbilden detekteras och man erhåller ett resultat som längst till höger i figur 5.[1]



Figur 5: Bilden som skickades in, bildens hough transform och detekterade linjer [7]

2.4 Programpaketet OpenCV

OpenCV är ett datorseendebibliotek som är open source-baserat och kan brukas inom många olika programeringsmiljöer såsom C++, Python, Java och MATLAB. OpenCV har även stöd för Windows, Linux, Mac OS och Andriod operativsystem. Väl etablerade företag som exempelvis Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda och Toyota nyttjar OpenCV:s bibliotek. OpenCV är alltså ett väl använt datorseendebibliotek inom industrin. Biblioteket består av mer än 2500 algoritmer vilka kan användas till att exempelvis i realtid detektera och känna igen ansikten, klassifiera objekt, följa kamerarörelse och följa rörliga objekt. OpenCV:s bibliotek fokuserar mestadels på datorseendeapplikationer som utförs i realtid [3].



3 RESULTAT

Detta kapitel beskriver hur datorseende kan användas för att navigera i ett vägnät. Kapitlet baserar sig på hur tidigare nämnda bildbehandlingsoperationer i den tekniska bakgrunden tillämpas och de krav som ställs på bildbehandlingen för navigering i ett vägnät.

3.1 Bildupplösning

Raspberry Pi kameran som används för vägnavigering har ett färdigt val av bildupplösningar. Om hela kamerans bildupplösning utnyttjas erhålls en upplösning på 3280×2464 . Figur 6 visar de olika upplösningsnivåerna som finns för Raspberry Pi kameran.

Då kameran används som hjälpmittel till att navigera i ett vägnät bör bildbehandlingstid inte spenderas på irrelevanta områden. Delar av bilden som inkluderar himlen, hustak och liknande likt figur 6 förkastas lämpligen för att göra datorseendet snabbare. Fokus bör istället begränsas till vägen och de väglinjer som söks. En större bild innehåller mer data som behöver behandlas vilket leder till längre beräkningstider. Skalas bildens upplösning ner till en lämplig nivå minskas både beräkningstiden och kravet på beräkningskraft.

Bildbehandlingen ska ske i realtid vilket medför att det är viktigt att optimera parametrar för att erhålla ett snabbare och mera responsivt system. Valet av bildupplösning sker experimentellt genom att prova olika upplösningar och utvärdera huruvida rätt upplösning är relevant för arbetsuppgiften att detektera väglinjer. I figur 6 ses att valet av en upplösning på 1640×922 erhåller en vid upplösning vilket fångar väglinjerna samtidigt som onödig data i vertikal-led minimeras.

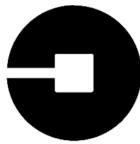


Figur 6: Möjliga bildupplösningar för Pi kameramodulen [8]

3.2 Bildbehandlingstid

Att undersöka hur lång tid bildbehanlingen tar är mycket svårt att estimera, experiment måste vanligtvis utföras. Raspberry Pi:n klockas i en viss frekvens men det är många processer som ska dela beräkningskraften, inte bara bildbehandlingen.

Däremot kan den totala tiden en viss process tar rent tidsmässigt undersökas. En enkel datorseendealgoritm har skrivits i C++ med OpenCV biblioteket, se appendix 6, för att undersöka detta. Algoritmen implementerar de olika



operationer och metoder som nämndes i den tekniska bakgrunden och appliceras på bilder för att undersöka hur beräkningstiden förändras.

Genom att applicera en timer på varje del av algoritmen för datorseende och applicera algoritmen på en bild undersöktes hur lång tid av den totala beräkningstiden algoritmens olika delar stod för. Tidtagning utfördes med hjälp av programmeringsspråket C++:s chronobibliotek med klocka av typen `high_resolution_clock`. Då systemet som testar algoritmen hade processer som kördes i bakgrunden varierar algoritmens beräkningstid något beroende på hur upptaget systemet är med andra processer. För att minimera felet som erhålls utfördes mätningen av beräkningstid 3 gånger och ett medelvärde för de erhållna tiderna beräknades. Medlevärdena för tiden av algoritmens olika delar samt den totala beräkningstiden finns presenterade i tabell 1.

Tidtagningen för algoritmen utfördes på en dator med följande specifikationer:

- Windows 10 Educational Operativsystem
- Intel Core i3-8100, 4 kärnor klockade på 3,6GHz, 6MB cache minne
- 16GB RAM

Tabell 1: Medelvärden av beräkningstider för algoritmens olika delmoment.

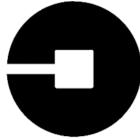
Delmoment	Tid (ms)
Maskning	0.300566
Gråskalning	31.046366
Lågpassfiltrering	40.885666
Kantdetektion	62.432733
Hough transform	28.517500
Total beräkningstid	163.182833

Som kan ses i tabellen tar delmomenten för lågpassfiltrering och kantdetektion längst tid. För att optimera den tid datorseendealgoritmen tar bör optimering av algoritmen lämpligvis fokusera på dessa delmoment då de är en större del av den totala tiden. Det är viktigt att nämna att den bild som används är av en verlig väg med alla störningar det medför. Den bil som ska konstrueras ska köra i en kontrollerad och mestadels svartvit miljö vilket bör medföra att ovanstående tider blir mindre.

Liknande bilder av olika upplösning användes för att estimera hur stor inverkan bildupplösningen har på det totala beräkningstiden. Mätningen av beräkningstiden utfördes på samma sätt som då beräkningstiden för algoritmens delmoment beräknades, mätningen utfördes 3 gånger på varje tid och ett medelvärde beräknades. Medelvärdena för beräkningstiderna av för de olika bildupplösningarna finns presenterade i tabell 2. Som kan ses tar större bilder längre tid att processera än mindre bilder, bildupplösningen bör alltså väljas så lågt som möjligt utan att påverka de tekniska egenskaperna i systemet.

Tabell 2: Medelvärden av beräkningstider för olika bildupplösningar.

Bildupplösning	Beräkningstid (ms)
1000x667	209.70933
788x525	139.16366
525x350	83.32360
263x175	56.24626

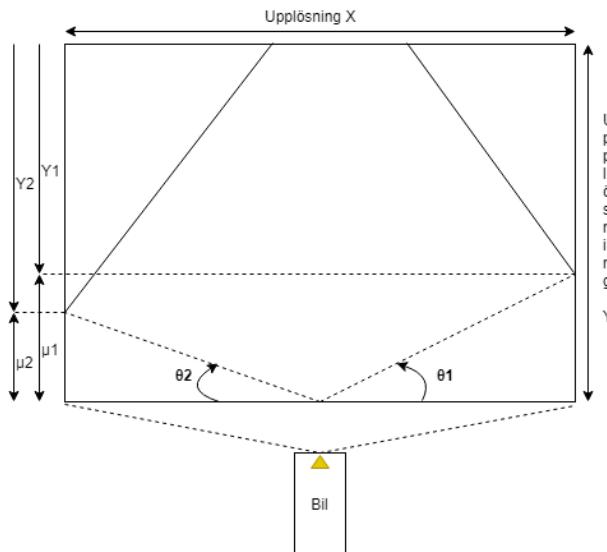


UNTER

Det projekt förstudien utförs inom sker i en kontrollerad miljö inomhus vilket ställer andra krav än om datorseende-algoritmen skulle appliceras i en icke-kontrollerad utomhusmiljö. En kontrollerad miljö ställer färre krav på algoritmen och delmoment som är nödvändiga i en icke-kontrollerad miljö kan möjligtvis uteslutas. Flera bilder på kontrollerade miljöer av vägar skickades till datorseendealgoritmen för att undersöka vilka linjer som detekterades. Delmoment av algoritmen, exempelvis lågpassfiltrering, uteslöts sedan och bilderna behandlades igen med denna förändrade algoritm. Bilderna före och efter förändringen jämfördes sedan för att undersöka vilken förändring som skett.

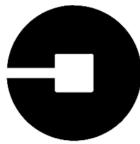
3.3 Bildberäkningar

I projektet förstudie skrivas inom ska datorseendet användas för att erhålla relevant information om bilens position och riktning i realtid. Informationen extraherad från datorseendebehandlade bilder ska skickas som in-parametrar till bilens reglersystem som med hjälp av denna data ser till att bilden håller sig i mitten av filen och inte slingrar sig framåt. För att regleringen ska ske så smidigt och exakt som möjligt är det viktigt att datorseendet ger precis och relevant data om bildens position och riktning. Datorseendebehandlade bilder kan ses som i figur 7, bilden har en vidvinkelkamera monterad centrerat längst fram som tar en bild av vägen. Datorseendealgoritmen identifierar linjerna för vägens filer och det är dessa linjer som används för beräkningarna.



Figur 7: Beräkningar för datorbehandlad bild

Om kameran är monterad centrerat längst fram på bilen och bilen kör i en fil så kan bilen ses som vara i mitten av bildens underkant och väglinjerna kommer in från bildens sidor. Koordinatsystemet för bilden har origo i det övre vänstra hörnet av bilden och detta medför att bilen bör ha koordinaten $(\frac{Upplösning X}{2}, Upplösning Y)$ och de två väglinjernas underkant bör ha koordinaterna $(Upplösning X, Y_1)$ respektive $(0, Y_2)$.



UNTER

Avståndet i Y-led från bildens underkant till då en väglinje kommer in från bildens sida kan fås genom att pixelvis stega upp från bilens underkant tills en pixel som ingår i väglinjen påträffas, det uppstegade avståndet bör då vara μ_1 respektive μ_2 för de respektive väglinjerna. Ekvation 4 beskriver hur vinklarna θ_1 och θ_2 kan beräknas

$$\theta = \arctan\left(\frac{\mu}{\frac{Upplösning_X}{2}}\right). \quad (4)$$

Med all relevant data känd kan två parametrar 5 6 tas fram för att beskriva bilens nuvarande position. Hur dessa parametrar kan användas för reglering av bildens position presenteras i tabell 3

$$\mu = \mu_1 - \mu_2 \quad (5)$$

$$\Theta = \theta_1 - \theta_2. \quad (6)$$

Tabell 3: Parametrar för navigering i vägnät med hjälp av datorseende.

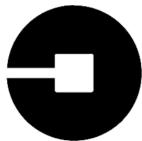
Parameter	Kör rakt	Sväng Höger	Sväng Vänster
μ	$\mu = 0$	$\mu < 0$	$\mu > 0$
Θ	$\Theta = 0$	$\Theta < 0$	$\Theta > 0$

4 SLUTSATS

Resultatdelen visar att valet av bildupplösning har stor påverkan på beräkningstiden. Det är därför en viktigt avvägning att välja en bildupplösning som snabbt kan processeras gentemot en stor bildupplösning som innehåller mer information. Genom att testa olika bildupplösningar kan en upplösningen som passar bäst för sitt specifika ändamål bestämmas. Valet av hur många bilder som ska behandlas per sekund bestäms uteslutande ifrån olika tester. Om den högsta upplösningen enligt tabell 2 används skulle det medföra att fem bilder per sekund kan beräknas. Låt säga att den autonoma taxibilen färdas med en hastighet på 1 m/s. Det skulle innebära att ny information finns tillgänglig var 20:e cm. Att var 20:e cm detektera nya väglinjer anses vara allt för lång tid. I scenariot då bilen färdas i 1m/s bör bildupplösningen minskas för att göra bildbehandlingen snabbare. Troligen bör väglinjerna uppdateras inom ett kortare intervall. Antalet bilder per sekund hänger således ihop med vilken bildupplösning man använder.

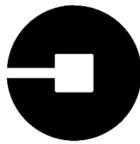
Datorseendealgoritmens mest beräkningsintensiva bildbehandlingsoperationerna är lågpassfiltrering och kantdetektion. Lågpassfiltrering är den brusreducerande filtrering som reducerar överflödigt brus i bilden och kan i kontrollerade miljöer möjligen ignoreras om erhållna bilder har få störmoment. Kantdetektionen tog längst tid att utföra och kan kräva viss optimering genom val av en annan kantdetektions algoritm eller genom att det kontrollerade testmiljön har få icke relevanta bildelement.

Bilden som erhålls från datorseendealgoritmen kan användas för att göra beräkningar om bilens relativa position i förhållande till väglinjerna. Genom att utföra beräkningar kan vinklar till väglinjer och avstånd bestämmas. Dessa parametrar som presenterades i tabell 3 skulle kunna användas av projektet i en regulator för att styra in bilen till mitten av vägfilen. Programpaketet opencv är det programpaket för datorseende som användes för att implementera datorseendealgoritmen i förstudiens resultatdel. Programpaketet har visat sig fungera väl och är ett mycket lämpligt färdigt programpaket för att utveckla datorseendealgoritmer.



5 REFERENSER

- [1] G. Bradski, A. Kaehler, B. Cambridge, Farnham, Köln, Sebastopol, Taipei, and Tokyo, *Learning OpenCV*, 2008. [Online]. Available: https://s3.amazonaws.com/academia.edu.documents/31785280/OREilly_Learning_OpenCV.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1550572168&Signature=NGHwpzJ4ucUBoPE2WfsU1K5BXxI%3D&response-content-disposition=inline%3B%20filename%3DLearning_Open_CV.pdf
- [2] R. Szeliski, *Computer vision : algorithms and applications*. Springer, 2011.
- [3] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [4] Wikipedia contributors, “Mask (computing) — Wikipedia, the free encyclopedia,” 2019, [Online; accessed 26-February-2019]. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Mask_\(computing\)&oldid=883299359](https://en.wikipedia.org/w/index.php?title=Mask_(computing)&oldid=883299359)
- [5] R. Krishna, “Computer vision: foundations and applications,” 2017.
- [6] A. Chahal, “DigitalCommons@USU In Situ Detection of Road Lanes Using Raspberry Pi,” Tech. Rep. [Online]. Available: <https://digitalcommons.usu.edu/etd>
- [7] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: <https://doi.org/10.7717/peerj.453>
- [8] Dave Jones, “picamera,” 2013-2019, [Online; accessed March 15, 2019]. [Online]. Available: https://picamera.readthedocs.io/en/release-1.12/_images/sensor_area_2.png



UNTER

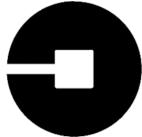
6 APENDIX

Listing 1: Datorseende med opencv

```
#include <opencv2/core.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include <chrono>
#include <iostream>

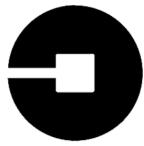
using namespace cv;
using namespace std;

int main(int argc, char** argv)
{
    //Declare output variables
    Mat dst, cdst, cdstP;
    //Load image
    Mat src = imread("Road1000x667.jpg");
    //Mask away top half
    auto start = std::chrono::system_clock::now();
    int cols = src.cols;
    int halfrows = src.rows/2;
    rectangle(src, Point(0, 0), Point(cols, halfrows), (0, 0, 0), FILLED);
    auto end = chrono::system_clock::now();
    chrono::duration<double> elapsed_seconds = end - start;
    cout << "Mask_time:" << elapsed_seconds.count() << "s\n";
    //Greyscale image
    start = std::chrono::system_clock::now();
    Mat greyMat;
    cvtColor(src, greyMat, COLOR_BGR2GRAY);
    end = chrono::system_clock::now();
    elapsed_seconds = end - start;
    cout << "Grey_time:" << elapsed_seconds.count() << "s\n";
    //Blur image
    start = std::chrono::system_clock::now();
    Mat blurMat;
    GaussianBlur(greyMat, blurMat, Size(5, 5), 0, 0);
    end = chrono::system_clock::now();
    elapsed_seconds = end - start;
    cout << "Blur_time:" << elapsed_seconds.count() << "s\n";
    //Edge detection
    start = std::chrono::system_clock::now();
    Mat edgeMat;
```



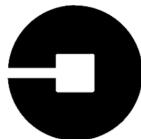
UNTER

```
Canny(blurMat, edgeMat, 50, 150, 3);
end = chrono::system_clock::now();
elapsed_seconds = end - start;
cout << "Edge_time:" << elapsed_seconds.count() << "s\n";
//Copy edges to the images that will display the results in BGR
cvtColor(edgeMat, cdst, COLOR_GRAY2BGR);
cdstP = cdst.clone();
//Standard Hough Line Transform
start = std::chrono::system_clock::now();
vector<Vec2f> lines; // will hold the results of the detection
HoughLines(edgeMat, lines, 1, CV_PI / 180, 150, 0, 0); // runs the actual detection
//Draw the lines
for (size_t i = 0; i < lines.size(); i++)
{
    float rho = lines[i][0], theta = lines[i][1];
    Point pt1, pt2;
    double a = cos(theta), b = sin(theta);
    double x0 = a * rho, y0 = b * rho;
    pt1.x = cvRound(x0 + 2500 * (-b));
    pt1.y = cvRound(y0 + 2500 * (a));
    pt2.x = cvRound(x0 - 2500 * (-b));
    pt2.y = cvRound(y0 - 2500 * (a));
    line(cdst, pt1, pt2, Scalar(0, 0, 255), 3, LINE_AA);
}
end = chrono::system_clock::now();
elapsed_seconds = end - start;
cout << "Hough_time:" << elapsed_seconds.count() << "s\n";
//Show results
imshow("Detected_Lines", cdst);
//Wait and Exit
waitKey();
return 0;
}
```



UNTER

I TEKNISK DOKUMENTATION



UNTER

Autonom taxibil

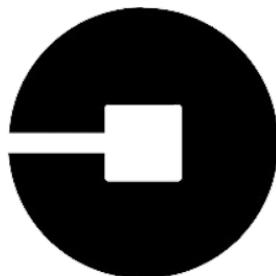
4 juni 2019

Teknisk dokumentation

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl

4 juni 2019

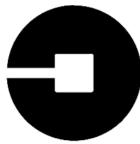
Version 1.0



UNTER

Status

Granskad	Grupp 9	2019-05-22
Godkänd		2019-xx-xx



UNTER

Autonom taxibil

4 juni 2019

Projektidentitet

Grupp E-post: jonca673@student.liu.se

Hemsida: <http://wwwisy.liu.se/edu/kurs/TSEA56/>

Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

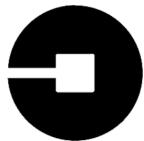
Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

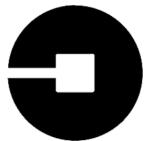
Namn	Ansvar	E-post
Jonathan Carlin	Projektleddare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Datorseendeansvarig (DAT)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se

**UNTER****INNEHÅLL**

1	Inledning	1
1.1	Bakgrund	1
1.2	Syfte	1
1.3	Avgränsningar	1
1.4	Struktur	1
2	Produktbeskrivning	1
3	Teori	2
3.1	USART busskommunikation	2
3.2	Näverksoptimering för navigation	2
3.3	Pulsbreddsmodulering	3
3.4	Datorseende	3
3.5	Hough transform	4
3.6	Ultraljudssensor för avståndsberäkning	5
3.7	Halleffektsswitch	5
4	Systemöversikt	5
4.1	Delsystemöversikt	6
4.2	Konstruktion	6
5	Delsystem - Kommunikationsmodul	7
5.1	USART busskommunikation	8
5.2	Kortastevägberäkning	8
5.3	Lagring av kördata	9
6	Delsystem - Styrmodul	9
6.1	Motor	10
6.2	Styrservo	11
6.3	Styrreglering	11
7	Delsystem - Sensormodul	12
7.1	Ultraljudssensor	13
7.2	Halleffektsswitch	14
7.3	Datorseende	15
8	Slutsatser	16
8.1	Slutsats Kommunikation	16
8.2	Slutsats Styrning	16
8.3	Slutsats Datorseende	16
8.4	Allmänna Slutsatser	16
A	Kravspecifikation	19
B	Banspecifikation och tävlingsregler	19
C	Systemskiss	19
D	Projektplan	19
E	Designspecifikation	19
F	Förstudie Kommunikation	19
G	Förstudie Styrning	19
H	Förstudie Sensor	19

**UNTER**

I	Användarmanual	19
J	Kopplingsschema Autonom Taxi	19
K	Programlistning	21



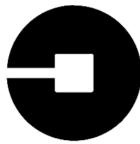
Autonom taxibil

4 juni 2019

UNTER

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
1.0	2019-05-22	Första version	Grupp 9	N.S.



UNTER

1 INLEDNING

I kandidatprojektet i Elektronik har en autonom taxibil som kan utföra köruppdrag konstruerats.

1.1 Bakgrund

Den tekniska dokumentationen är tänkt att agera som ett konstruktionsunderlag för någon som vill konstruera en liknande autonom taxibil. Dokumentationen lämpar sig till läsare med grundkunskaperna för kursen TSEA56.

1.2 Syfte

Syftet med detta dokument är att beskriva konstruktionen av den autonoma taxibil som konstruerats under utförandet av kursen kandidatprojekt i Elektronik (TSEA56) vid Linköpings Universitet. Dokumentet går igenom hur bilen har konstruerats och har detaljerade beskrivningar för hur bilen fungerar.

1.3 Avgränsningar

Rapporten begränsar sig till att fokusera på den färdiga taxibilen. Rapporten kommer att beskriva de tekniska stegen som valdes för att konstruera taxibilen. Rapporten kommer inte att ta i beaktning hur andra möjliga lösningar hade kunnat implementerats utan fokusera på de designval som gjordes under projektet.

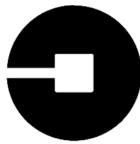
1.4 Struktur

Dokumentet är uppdelat i två huvuddelar. Den första delen beskriver teorin bakom projektet och den andra delen beskriver bilens konstruktion.

2 PRODUKTBeskrivning

Produkten är en autonom taxibil bestående av ett chassi med fyra hjul, en motor, ett styrservo, sensorer, en enkortsdator, en kamera och ett virkort med styrelektronik.

Bilen är tänkt att navigera i ett vägnät för att utföra köruppdrag där bilen tar in ett vägnät med ett antal angivna destinationer och sedan autonomt besöker alla angivna destinationer på kortast möjliga väg. För att utföra ett köruppdrag placeras bilen på en position i vägnätet och denna position anges som bilens startposition. Därefter anges ett antal destinationer för bilen att besöka. Angivna destinationer representeras i vägnätet alltid av en stopplinje och vägnätets väglinjer består av svarta heldragna sträck för kanter med svarta streckade sträck för vägens mittlinje. I vägnätet kan det förekomma hinder i form av tredimensionella objekt som bilen stannar framför tills hindret avlägsnas.



UNTER

3 TEORI

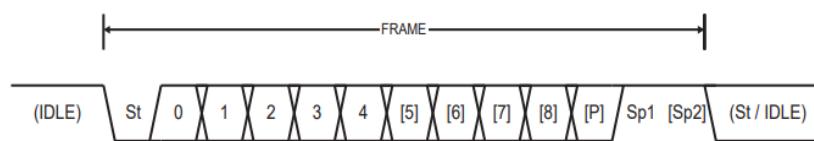
Projektet använder sig av olika sensorer, algoritmer och metoder för att utföra autonoma köruppdrag. I följande rubriker presenteras den bakomliggande teorin som gjorde projektet möjligt.

3.1 USART busskommunikation

Kommunikationen mellan Raspberry Pi (RPI) och övriga modulers processorer (två AVR-processorer) sker via USART. USART är ett seriellt gränssnitt som kan programmeras till att kommunicera asynkront och till skillnad från UART även synkront. Det finns flertalet olika protokoll för kommunikation via USART och i detta dokument beskrivs detaljerna för det protokoll som används av AVR ATmega1284P-processorn. På AVR ATmega1284P-processorn finns två USARTs, USART0 och USART1, endast USART0 ska användas på båda processorer. På processorna är pinnarna PD0 och PD1:s alternativa funktioner RXD0 och TXD0, vilka används för att mottaga och sända data via USART0 respektive. USART stödjer fyra olika lägen: Normal asynchronous, Double Speed asynchronous, Master synchronous and Slave synchronous. USART0 ska endast användas till asynkron kommunikation, vilket betyder att en extern synkron klocka inte behöver användas. I detta fall beter sig bussen som en UART.

Kommunikationen mellan enheter upprättas genom att man förbestämmer en vald *baud rate*, baud rate:n beskriver kommunikationshastigheten i bitar per sekund på bussen. En förbestämd klockfrekvens väljs även på processorns systemklocka detta då den används för att sampla TXD0 och RXD0 i precisa intervaller för att kunna uppehålla kommunikationen.

Ett seriellt frame format ska definieras för att överföra information på bussen. ATmega1284P-processorn stödjer 30 olika kombinationer av startbit, databitar, paritetsbitar och stoppbitar, se figur 1.

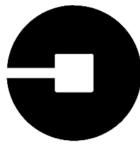


Figur 1: Möjliga kombinationer av frame-format för ATmega1284P. Bitar i hakparanteser är valfria.

3.2 Näverksoptimering för navigation

INom grafteorin brukar man beskriva nätverk som en graf. En graf består av ett antal noder och bågar som förbinder noderna. En väg är en sekvens av bågar som man kan följa från start till slut.

En viktad, riktad graf är en effektiv representation av ett stadsvägnät. Beroende på vilka villkor man ställer för en potentiell väg så finns det olika algoritmer för att beräkna den snabbaste vägen. Problemet att minimera vägen från en nod till en annan kan lösas med hjälp av dijkstras algoritm [1].



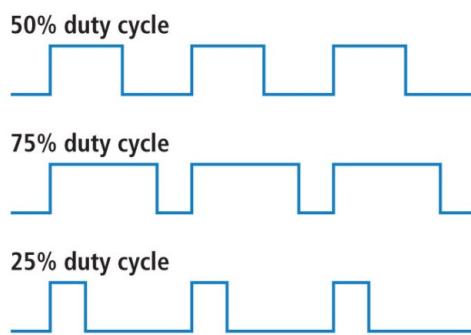
UNTER

3.3 Pulsbreddsmodulering

Pulsbreddsmodulering, eller pulse width modulation (PWM) på engelska, innebär att man bestämmer under hur lång tid en signal ska vara hög eller låg. Det är väldigt användbart vid styrning av ett servo eller en motor då man kan styra insignalen på ett mycket mer exakt sätt än genom att skicka in en signal med olika stora spänningar. Detta kan beskrivas med hjälp av en såkallad *Duty Cycle*. En duty cycle är en procentsats som beskriver hur länge en puls är hög under en period och beräknas med ekvation 1

$$\text{Dutycycle} = \frac{p}{T} \quad (1)$$

där p är pulsens bredd (ms) och T är periodtiden (ms). Se figur 2 för en grafisk beskrivning.



Figur 2: Beskrivning av "Duty Cycles"

De flesta mikrodatorer har en eller flera inbyggda pulsbreddsomvandlare och för att komma åt dem måste man använda någon form av klocka (i projektet användes de två 16-bitars klockorna Timer/Counter1 och Timer/Counter3). För att bestämma den önskade periodtiden på signalen behöver man mikrodatorns klockfrekvens (f_{clock}). Med hjälp av den kan man beräkna hur många klockcykler som körs under den önskade periodtiden (f_{pulse}) och utifrån ekvation 2 kan man beräkna antalet klockcykler ($Ticks$)

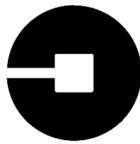
$$Ticks = \frac{f_{clock}}{f_{pulse}} - 1. \quad (2)$$

3.4 Datorseende

Datorseende innebär att bild eller video kan tolkas av ett datorprogram för att dra slutsatser om vad bilden innehåller. En dator tolkar bilder numerisk där nummer representerar egenskaper hos bilden. Vad numren representerar beror på bildens färgsystem, alltså det sätt som bilden är representerad. Ett välkänt färgsystem är RGB i vilket bilden representeras av tre färgkanaler, ett nummer för rött, ett för grönt och ett för blått. För att dra slutsatser om vad bilden innehåller utför datorseendealgoritmer bildbehandling med olika operationer

3.4.1 CEILAB färgsystem

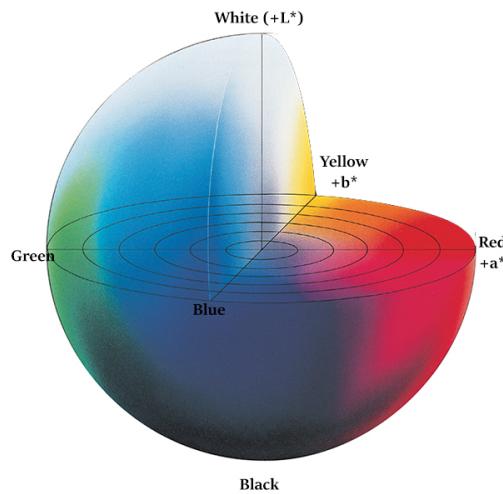
Färgsystemet CEILAB är ett välanvänt färgsystem för detektion av färger. Färgsystemet har tre parametrar, L, a, och b, som spänner upp systemets färgrymd som sett i figur 3.



UNTER

Autonom taxibil

4 juni 2019



Figur 3: Bild av färgrymden för färgsystemet CEILAB.

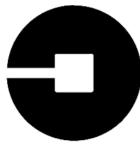
L-parametern betecknar ljusintensiteten i färgen medan a- och b-parametrarna beskriver färginnehållet. Parameter a betecknar för positiva värden mängden rött och för negativa värden mängden grönt i färgen medan b-parametern för positiva tal betecknar mängden blått och för negativa tal mängden gult i färgen.

3.5 Hough transform

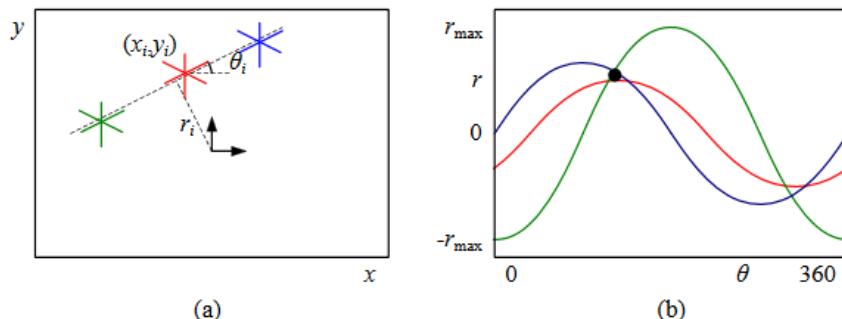
Hough transformen är en metod för att detektera enkla strukturer som exempelvis linjer i bilder. Transformen utförs på binära bilder där färgrymden endast består av två värden, vitt eller svart, och tillämpas lämpligen på en bild som genomgått kantdetektion. Metoden baserar sig på att en linje i det kartesiska koordinatsystemet kan parametriseras i polära koordinater [3](#)

$$r = x \cos \theta + y \sin \theta \quad (3)$$

där r är det vinkelräta avståndet från origo till linjen och θ är vinkeln mellan linjen och en horisontell linje taget moturs. Hough transformen utförs genom att för varje punkt i det kartesiska koordinatsystemet ta fram en mängd linjer som inkluderar den givna punkten. Varje linje parametriseras till det polära koordinatsystemet där en linje representeras av en punkt. Utförs detta för alla linjer som passerar genom punkten erhålls en stor mängd punkter i det polära koordinatsystemet och punkterna bildar en vågliknande struktur som i figur [4](#).



UNTER



Figur 4: (a) Tre punkter i ett kartesiskt koordinatsystem. (b) De tre punkternas Hough transform[2].

I figur 4 finns det tre punkter i kartesiska koordinater och dessa tre punkter har respektive Hough transform i det polära koordinatsystemet. I det område där flest linjer skär varandra, som är markerat med en svart punkt, är det störst sannolikhet att en linje som inkluderar alla punkterna finns. Den svarta punkten är en linje parametrerad på polär form och är den sträckade svarta linjen i det kartesiska koordinatsystemet.

3.6 Ultraljudssensor för avståndsberäkning

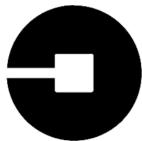
Ultraljudssensorn SRF04 är en avståndssensor som med hjälp av ultraljud mäter avstånd till objekt. Genom att skicka ut en ljudvåg som färdas i ljudets hastighet för att sedan reflekteras tillbaka till sensorn när den träffat ett objekt kan avståndet till objektet beräknas. Med kännedom om vilket medium ljudvägen färdas genom och tiden det tog att detektera den reflekterade vågen sedan ljudet skickades ut kan avståndet till objektet avgöras med hög precision.

3.7 Halleffektsensor

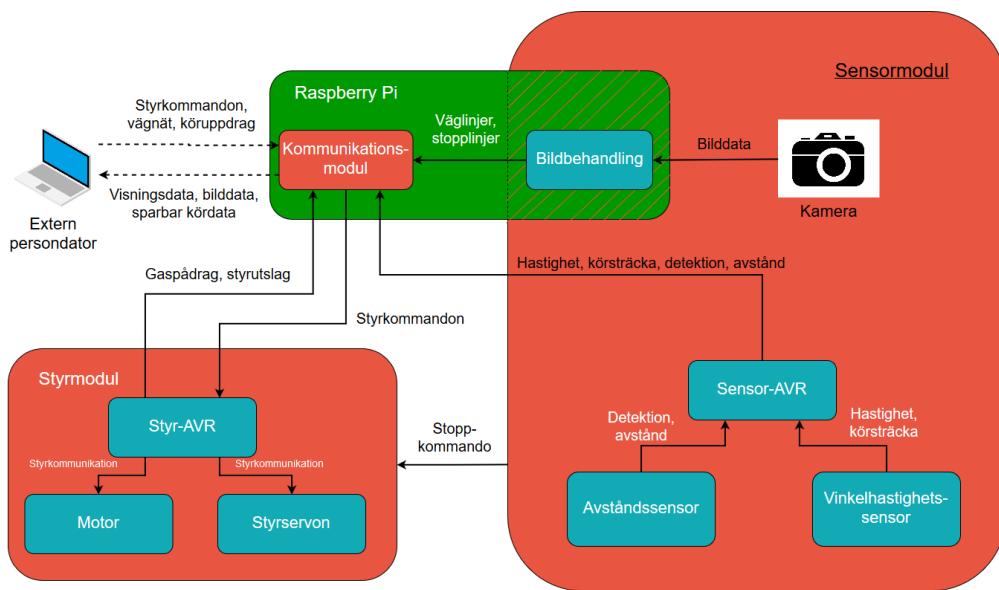
En halleffektsensor är en elektrisk komponent bestående av en tunn metallplatta som en ström passerar genom. Då metallplattan är i närbeten av ett elektrisk fält avleds elektronerna till ena sidan av metallplattan och detta medför att en potentialskillnad uppstår utmed sidan av metallplattan. Potentialskillnaden detekteras av halleffektsensoren som ger en hög signal tills dess att metallplattan inte längre är i närbeten av ett magnetfält.

4 SYSTEMÖVERSIKT

Bilen består av tre olika delsystem: En kommunikationsmodul, en sensormodul och en styrmodul. Sensormodulen ansvarade för att inhämta mätdata från bilens olika sensorer. Styrmodulen utnyttjade mätdata från sensormodulen för att reglera bilen under körning. Kommunikationsmodulen ansvarade för att upprätthålla kommunikationen med persondatorn samt den interna dataöverföringen mellan styrmodulen och sensormodulen. Utöver dessa uppgifter ansvarade även kommunikationsmodulen för kortaste-väg-beräkningar.



UNTER



Figur 5: Översikt av systemet.

4.1 Delsystemöversikt

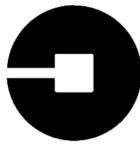
Kommunikationsmodulen låg på RPI:n och var den beslutande modulen. Kommunikationsmodulen komunicerade både med användaren och de andra modulerna. Användaren kunde ge kommunikationsmodulen direktiv som exempelvis körkommandon. Modulen förmedlade även direktiv till de andra delsystemen, exempelvis styrutslag till styrmodulen.

Styrmodulen låg på en ATmega1284P och var det system som var närmast kopplat till bilens motor och styrning. Styrmodulen tog emot styrkommandon från kommunikationsmodulen via UART-kommunikation. Väl på mikrodatorn tolkades styrkommandona om till pulsredder som sedan sändes ut till motorn respektive styrningen.

Sensormodulen var det tredje delsystemet i bilen och tog hand om både datorseende och sensordata. Datorseendet låg på RPI:n och ansvarde för att detektera väglinjerna. Utifrån datorseendet tog sedan kommunikationsmodulen beslut över vad som skulle göras i form av styrregleringar. Delsystemet innehöll även en ATmega1284P och hade två uppgifter, att detektera ifall det fanns något hinder på vägen och beräkning av hur långt bilden färdats.

4.2 Konstruktion

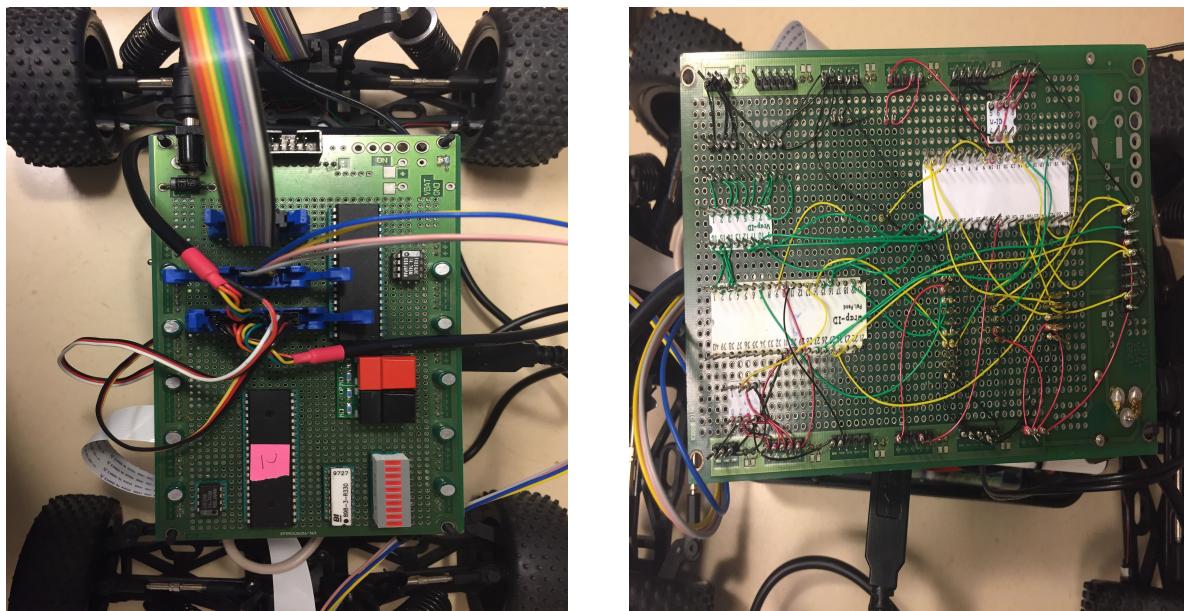
Delsystemen konstruerades parallellt under projektet för att på så vis kunna testas individuellt innan de slutligen kopplades samman. Styr och sensormodulen konstruerades på var sitt virkort och testades kontinuerligt innan de kopplades ihop. Detta kunde ske i flera steg t.ex som sensormodulen som först programmerades för ultraljudet, sedan för halsensorerna, sedan kopplades dessa samman innan de monterades på bilen.



UNTER

Autonom taxibil

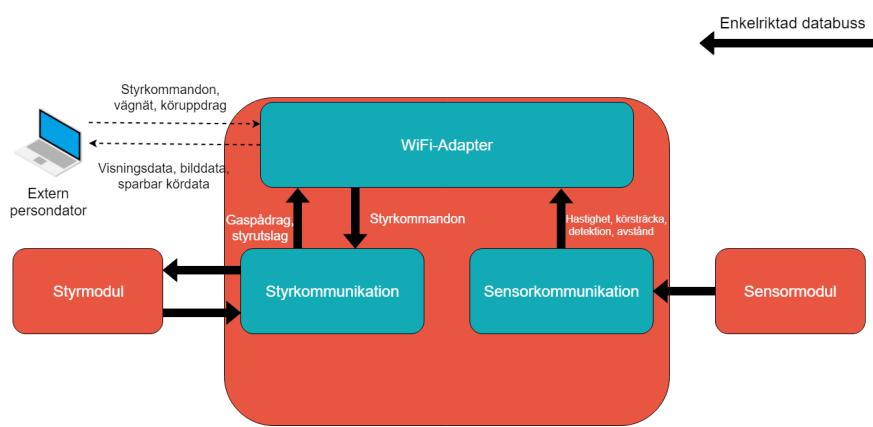
4 juni 2019



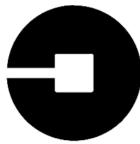
Figur 6: Virkort som användes i projektet.

5 DELSYSTEM - KOMMUNIKATIONSMODUL

Kommunikationsmodulens resurs var en raspberry pi 3 (RPI). Den tolkade input från användaren för att avgöra vilket körläge som skulle användas. För kommunikationen med användaren utnyttjades RPI:ns inbyggda wifi-adapter. De andra modulerna ställdes in efter körläge via busskommunikation från kommunikationsmodulen. I manuellt läge avaktiverades sensormodulens bildbehandling och styrmodulens reglering. Det semiautonoma och autonoma körlägena använde olika metoder för att ta fram en körrutt. Kommunikationsmodulen användes för att avgöra vilket metod som skulle användas.



Figur 7: Översikt av kommunikationsmodulen.



UNTER

5.1 USART busskommunikation

På ATMega1284P-processornerna är pinnarna PD0 och PD1:s alternativa funktioner RXD0 och TXD0, vilka används för att mottaga och sända data via USART0 respektive. Dessa alternativa funktioner för pinnarna aktiveras genom att sätta korresponderande bitar RXEN0 och TXEN0 i kontrollregistret UCSR0B till 1. Då USART0 användes som en UART valdes läget normal asynchronous bland de fyra möjliga lägena. Normalt asynkront läge väljs genom att sätta UMSEL00-biten i kontrollregistret UCSR0C till 0 och U2X0-biten i kontrollregistret UCSR0A till 0, vilket är bitarnas standardinställningar. UMSEL00 styr huruvida USART0 är asynkron eller synkron och U2X0 styr huruvida USART0 är i läget Normal asynchronous eller Double Speed asynchronous. Baud rate:n valdes till 9600 bps och klockfrekvensen till 1.8432 MHz efter en tabell i ATMega1284P databladet av vanliga baud rate inställningar [3]. Dessa valda värden garanterar ett fel på 0.0% enligt tabellen.

För att erhålla den önskade baud rate:n för USART0 ställs baud rate registret UBRR0 in. Ekvation 4 beskriver hur detta värde tas fram.

$$UBRR0 = \frac{F_{CPU}}{16BAUD} - 1 \quad (4)$$

Detta värde finner man i processorns datablad för vanliga inställningar på baud rate men genom att tillämpa ekvation 4 implementerades den som ett makro och således behövdes inte det svårtolkade UBRR-värdet hårdkodas in i källkoden. Koden som genererar UBRR0 är som följer

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / 16) + (USART_BAUDRATE / 2)) / (USART_BAUDRATE )) - 1
```

Makrot matchar inte ekvationen 4 helt och hållet, detta eftersom denna variant avrundar UBRR0 på ett sådant sätt som säkerställer att baud rate:n blir så nära det önskade värdet som möjligt. Frame formatet som ska användas bestod av en startbit, 8 databitar och en stoppbil. Antalet databitar i en frame bestäms av UCSZ01:0-bitarna i kontrollregistret UCSR0C och UCSZ02-biten i UCSR0B. Antalet stoppbitar bestäms av USBS0-biten och paritetsläget bestäms av bitarna UPM01:0 i kontrollregistret UCSR0C. En komplett initialisering av USART0 visas i koden 1.

```
#include <avr /io.h>

#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / 16) + (USART_BAUDRATE / 2)) / (USART_BAUDRATE )) - 1

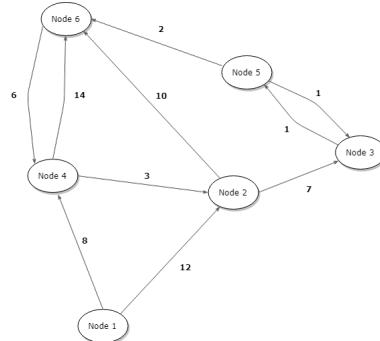
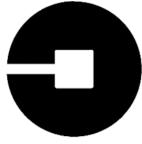
void USART_init()
{
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);      // Turn on the transmission and reception circuitry
    UCSR0C = (1 << UCSZ00) | (1 << UCSZ01); // Set frame format: 8 data bits, 1 stop bit

    // Set the baud rate registers
    UBRR0H = (BAUD_PRESCALE >> 8); // Load upper 8- bits of the baud rate value into UBRR0H
    UBRR0L = BAUD_PRESCALE;        // Load lower 8- bits of the baud rate value into UBRR0L
}
```

Listing 1: Initialisering av USART0 på en ATmega1284P.

5.2 Kortastvägberäkning

För att bilen skulle ta fram en rutt vid ett autonomt köruppdrag så krävdes någon form av vägberäkning. Denna beräkning utnyttjade dijkstras algoritm och vägnätets grafrepresentation. Se figur 8 för en visuell representation av en graf.



Figur 8: Riktad och viktad graf.

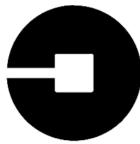
Ett köruppdrag bestod av en startnod och ett okänt antal destinationsnoder. För att ta fram en kort väg som besökte alla destinationer utfördes dijkstras algoritm iterativt för att jämföra avståndet till alla destinationer. Sedan valdes den kortaste rutten och motsvarande destination användes som startnod i nästa iteration av dijkstras. Denna iterativa algoritm gav en sekvens med bågar som blev bilens körrutt, men det är inte en optimal lösning på nätverksproblem. Att lösa detta problem optimalt skulle kräva en brute force-metod, vilket gör den ovan nämnda algoritmen mer skalbar (dock med försämrat resultat).

5.3 Lagring av kördata

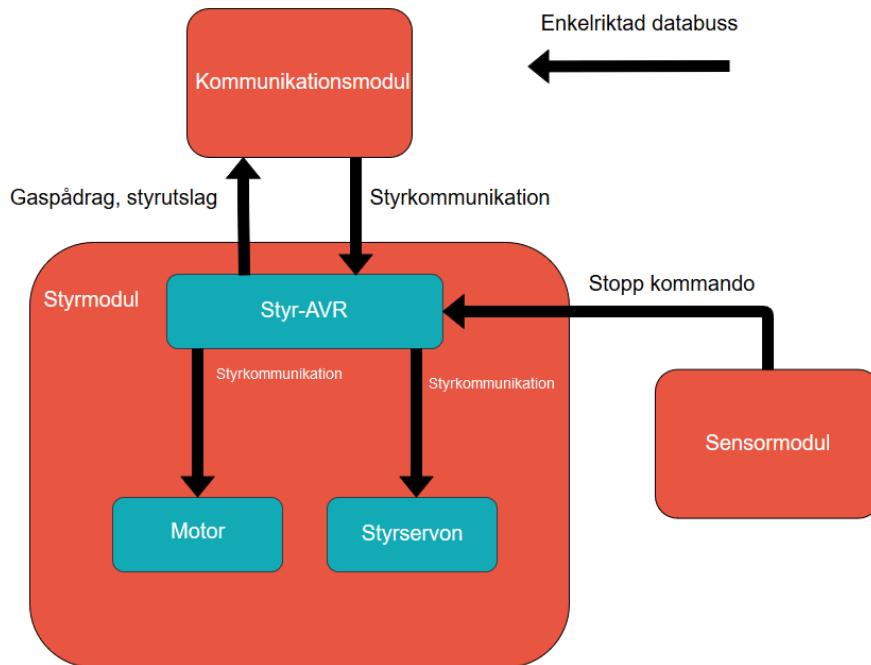
Det kan vara intressant att i utvecklingssyfte studera kördata. Kördata såsom körtid, körd sträcka, gaspådrag och styrutslag valdes att sparas i ett csv-format. Csv-formatet gör att datan enkelt i både exempelvis MATLAB eller Microsoft Excel kan öppnas och analyseras i form av grafer.

6 DELSYSTEM - STYRMODUL

Denna rubrik beskriver detaljerat styrmoden på bilen som hanterar all motor och servostyrningen för blien.. Koden för att reglera pulsbreddmoduleringen för motorn och styrservot skrevs helt i C kod och programmerades in på en enkortsdator med hjälp av en JTAG. Regleringsalgoritmen skrevs i C++ och skrevs delvis på kommunikationsmodulen i RPI:n. Kommunikationen mellan dem skedde med en UART kabel. I figur 9 visas en grafisk översikt av delsystemet.



UNTER



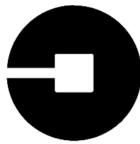
Figur 9: Grafisk beskrivning av styrmodulen.

6.1 Motor

Motorn som användes på bilen ingick i den robotplattform gruppen blev tilldelad. Inga förändringar eller uppdateringar fick göras med hårdvaran, gruppens uppgift var istället att koppla upp motorn till styrmodulen via ett virkort och sedan programmera en enkortsdator (ATmega1284P) för att styra den. Detta gjordes genom att använda en pulsbreddsomvandlare och genom att skicka in olika breda pulser under en periodtid kunde hastigheten regleras. En 16-bitars klocka användes för att reglera pulsbredden och *Fast PWM* var den inställning som användes. Motorn kunde hantera en maximal pulsbredd på 2,0 ms och en minimal pulsbredd på 1,0 ms. Med dessa gränser kände gruppen att det var naturligt att använda 1,5 ms som standardvärdet för motorervot och det motsvarade i praktiken "stå still". Om pulsbredden sattes till ett större värde än 1,5 så körde bilen framåt och om pulsbredden sattes till ett mindre värde så backade bilen. Mikrokontrollern använde en klockfrekvens på $f_{clock} = 1.8432MHz$ så för att ta fram en periodtid på 20 ms togs antalet klockcykler (*Ticks*) under den tiden fram med hjälp av ekvation 2. Antalet ticks under 20 ms beräknades genom att dividera servots frekvens (50 Hz) med klockfrekvensen som i ekvation 5.

$$Ticks = \frac{1.8432 * 10^6}{50} - 1 = 36863. \quad (5)$$

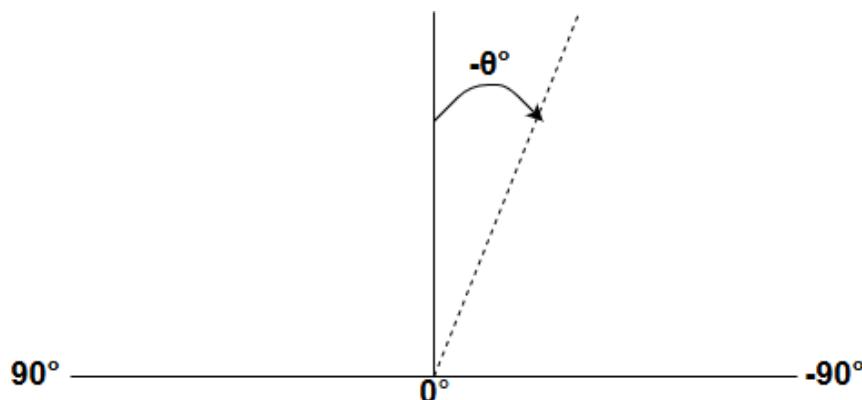
Då 16-bitars klockan användes så behövdes ingen prescaling ($36863 < 65535$). Den maximala pulsbredden har en duty-cycle på 10% ($2/20 = 0,1$) och den minimala pulsbredden har en duty-cycle på 5% ($1/20 = 0,05$). Med dessa procentsatser var det enkelt att ta reda på att antalet ticks utifrån , standardvärdet var cirka 2765, maxvärdet var cirka 3686 och minvärdet var cirka 1843.



UNTER

6.2 Styrservo

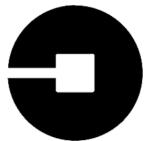
Styrservot fungerade på samma sätt som motorn. Genom att skicka in en pulsbredd mellan 1,0 ms och 2,0 ms kunde man reglera styrutslaget på servot. 1,5 ms användes även här som standardvärde för det tänkta styrutslaget på 0° , se figur 10. Det visade sig dock tidigt inte stämma då standardvärdet på styrservot innebar att det hade ett mindre utslag åt höger. Detta korrigerades genom att testa olika värden på styrservot tills ett värde som gav ett styrutslag på ca; 0° hittades (cirka 2515). För vara säkra på att gruppen inte av misstag skulle skicka in en pulsbredd som var större eller mindre än maxvärdet respektive minvärdet (och potentiellt skada servot) implementerade gruppen en "checkfunktion" som kontrollerade att värdet som servot skulle sättas till var inom gränserna. Detta var framförallt användbart vid testandet av styralgoritmen då den hade ibland skickade ut ett oönskat värde som kunde ligga utanför gränsen.



Figur 10: Beskrivning av styrutslag.

6.3 Styrreglering

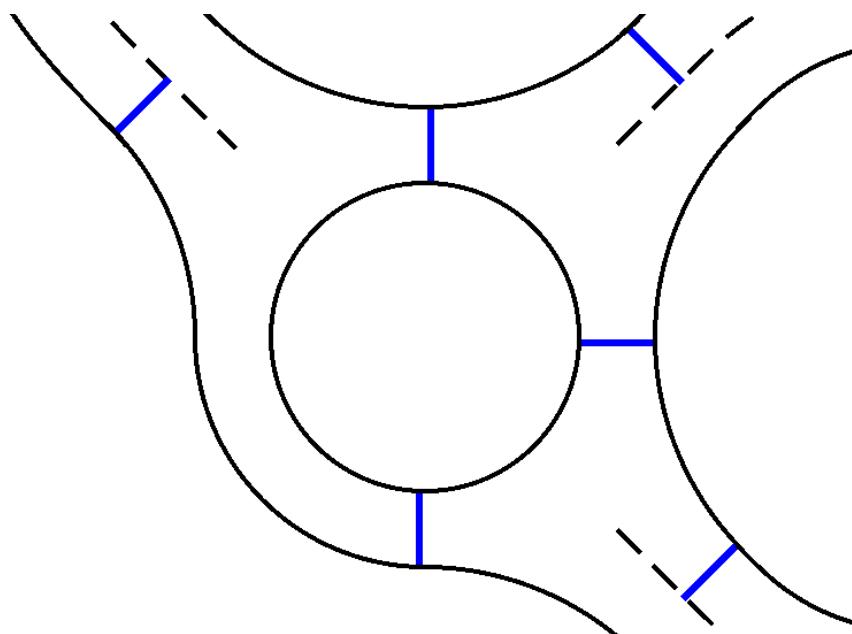
Styrregleringen implementerades på RPI:n och fungerade på olika sätt för *Manuellt* läge och *Autonomt/Semi-autonomt* läge. All styrning initierades från RPI:n i terminalen. I manuellt läge styrde användaren själv bilen med tangenterna *W*, *A*, *S*, *D* och *SPACE*, *W*: kör framåt, *A*: sväng väsnter, *S*: stanna/bakcka, *D*: sväng höger och *SPACE*: ställ hjulen rakt. En tryckning på en *W* ökade eller minskade pulsbredden som skickades till motorn med ett konstant värde på 10 ticks och en tryckning på *A/D* ökade respektive minskade värdet på pulsbredden med 90 ticks. Detta gjorde att styrregleringen i manuellt läge fungerade men kunde uppfattas som något "hackig". Autonomt och semi-autonomt läge använder datorseendet för att reglera styrningen. Bilens användare tre olika sorters regleringar beroende var i banan den befinner sig, dessa var *control_road*, *control_entry* och *control_rounabout*. Anledningen till att tre olika regleringsalgoritmer användes var för att standardregleringen var beroende av datorseendet som tog fram en referenslinje utifrån de två väglinjer som fanns på banan. Då bilen körde in i en rondell med standardreglering tappade den båda linjer vilket resulterade i att den körde av vägen. För att undvika detta programmerades bilen till att följa den högra linjen in i en rondell (*control_entry*), sedan byta till att följa den vänstra linjen under tiden bilen befann sig i rondellen (*control_rounabout*) och till sist byta tillbaka till att följa högerlinjen vid utkörning från rondellen (*control_entry*). Efter att bilen kört cirka 170 centimeter med *control_entry* reglering så bytte den tillbaka till normal reglering (*control_road*). I figur 11 presenteras ett exempel på en rondell som gruppen använde under projektet.



UNTER

Autonom taxibil

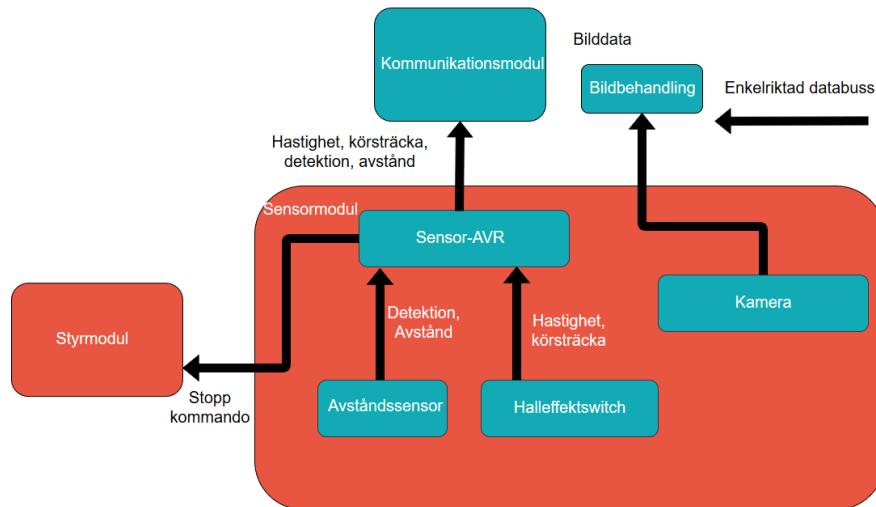
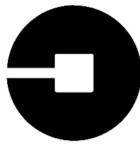
4 juni 2019



Figur 11: Projicerad rondell som gruppen använde under projektet.

7 DELSYSTEM - SENZORMODUL

Senzormodulen var den modul som ansvarade för bilens sensorer. Senzormodulen bestod av en ATmega1284p och en Raspberry Pi Camera Module v2. Senzormodulens Atmega1284p kom att under projektets gång refereras som sensorAVR.

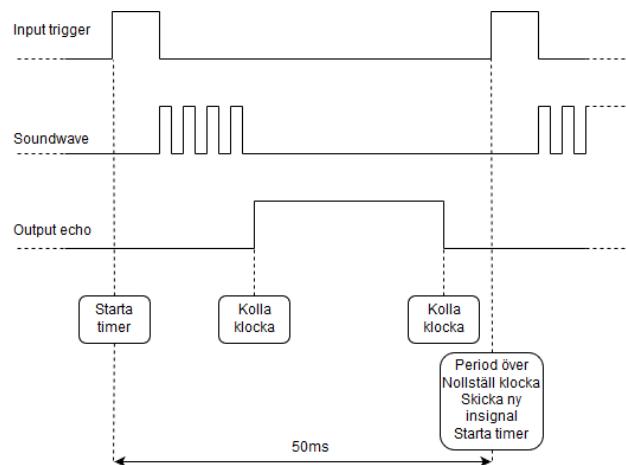
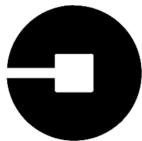


Figur 12: Översikt av sensormodulen.

7.1 Ultraljudssensor

För hinderdetektion aktiverades ultraljudssensorn genom att man sände ut en pulsbredmodulering till sensorAVR:n. För att hela händelseförlloppet, som visas i figur 13, skulle kunna genomföras valdes en pulsbredd på 50 ms. Pulsbreddsmoduleringen valdes att placeras på en 16-bitars klocka och förskalades med värdet 8. För att initiera ultraljudssensorn behövdes en insignal på minst 10 μ s skickas till sensorn. En pulsbredd på 2.5 ms valdes som initieringsignal och gav upphov till en duty cycle på 5% vilket ansågs vara tillräckligt liten för att inte störa resterande händelsförlopp.

När väl ultraljudssensorn hade startats sände den ut en ljudvåg och skickade även en hög utsignal tillbaka till sensoravr:n. Ultraljudssensorn höll en hög utsignal ända fram tills att ljudet hade studsat tillbaka till sensorn. Genom att klocka tiden mellan att utsignalen sändes ut som hög till att den sändes ut som låg kunde avståndet beräknas.

**Figur 13:** Händelseförlopp ultraljudssensor.

Beräkningen av avståndet skedde på mikrodatorn och gick till genom att generera avbrott utifrån positiv respektive negativ flank från ultraljudssensors utsignal. Mikroprocessorn tog emot en positiv flank och triggade ett avbrott. Avbrottsrutinen ändrade villkoret att ett avbrott skulle genereras när insignalen från ultraljudssensorn fick negativ flank. Avbrottsrutinen nollställdে även räknaren TCNT1 så att räknaren skulle räkna tiden till nästa avbrott. När nästa avbrott inträffade ändrade avbrottsrutinen att nästa avbrott skulle reagera på positiv flank. Tiden hämtades från ICR1 där värdet från den då nollställda TCNT1 hade räknats upp till fram till den negativa flaken. Sålunda kunde tiden det hade tagit för ljudvågen att färdas till objekt och tillbaka bestämmas. Med hjälp av ekvation 6 kunde avståndet beräknas

$$\text{totaldistance} = \frac{343 \cdot 8 \cdot \text{totaltime}}{2 \cdot f_{clock}}. \quad (6)$$

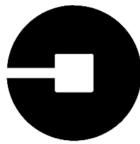
Variablen totaltime tog in det lagrade klockvädet från ICR1. Värdet baserades på antal "ticks" och får att få fram den faktiska tiden det tog behövde man ta hänsyn till mikrodatorns klockfrekvens samt förskalningsfaktorn. Avståndet kom även att divideras med två för att ta i beaktning att ljudvågen hade färdats fram och tillbaka.

7.2 Halleffektswitch

Jämst fördelade runt hjulen på bilen var tio magnetar placerade. När hjulet roterade passerades magneterna och halleffektswitchen detekterade dess potentialskillnad. Körd sträcka kunde således beräknas genom att ha vetskaps om hjulradien samt antal passerade magnetar. Detta gjordes möjligt genom att låta Counter0 på sensorAVRn agera som en räknare och räkna upp varje gång halleffektswitchen sände en hög signal. OCR0A registret ställdes in att ställa in hur många gånger räknaren skulle räkna upp innan overflow. Timer Interrupt Mask Register (TIMSK0) konfigurerades att utlösa ett avbrott varje gång räknaren fick overflow. Väl i avbrottsrutinen kunde den totala kördan sträckan beräknas 7,

$$\text{total_distance_hal} = \frac{\text{wheel_circumference}}{10} \cdot \text{OCR0A}. \quad (7)$$

där total_distance_hal var den totalt uppmätta kördan sträckan. OCR0A ställdes in att räkna upp en gång innan overflow. Detta gav en precision på $\frac{\text{wheel_circumference}}{10} = 2.6$ cm för bilens kördan sträckan, då hjulomkretsen hade beräknats till 26 cm.



7.3 Datorseende

Bilens datorseende använder sig av en Raspberry Pi kameramodul version 2 som är monterad fram på bilen. Kamerans position är 16 cm ovanför vägen och vinklad neråt så att så stor del av vägen som möjligt fångas in samtidig som störningar i bilderna från bilens hjul och skuggor samt väggarna av rummet minimeras. På kameran är en lins av fisheye typ monterad, kameran är således en vidvinkel kamera med en synvinkel på 160°.

Datorseendealgoritmen är skriven i programmeringsspråket C++ och använder sig av programbiblioteket OpenCV för att utföra bildbehandlings operationer. För att ta bilder med kameran använder sig datorseendet av programbiblioteket RaspiCam som i sin tur använder sig av C-biblioteket MMAL, Multimedia Abstraction Layer, för att styra kameramodulens hårdvara.

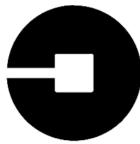
Datorseendet påbörjas med att en videoström för kameramodulen öppnas och parametervärden för kamerans bildupplösning och shutterspeed sätts. Bildupplösningen för kameran är satt till en bredd på 320 pixlar och en höjd på 240 pixlar. Upplösningen valdes liten då bildbehandlingsoperationer har en tidskomplexitet som växer i förhållande till bildstorlek. Datorseendets uppgift är att förse bilens reglersystem med parametrar för bilens position och behöver då utföras så fort som möjligt. Kamerans shutterspeed är inställt på 12 på grund av störningar introducerade från de projektorer som projicerar vägnätet bilden färdas på. Att vägnätet är projicerat medför att de bilder kameran tar har stora störningar i form av gula linjer som går rakt över bilden. Olika shutterspeeds testades och en shutterspeed på 12 resulterade i att störningen av gula linjer minimerades.

När en bild har hämtats från kameran utför datorseendet ett antal bildbehandlingsoperationer för att detektera väglinjer i bilden. Först maskas bilen med en mask vald sådan att de störningar som alltid uppstår i uppkant av bilden filtreras bort. Kamerans position och vinkel medför att en del av bilens hjul syns i nedre kanten av bilden så även denna störning maskas bort.

Efter det att maskning har utförts förs bilden, som av kameran togs på BGR format, en färgrymd där tre värden representerar färger i en andel blått, grönt och rött, till färgsystemet CEILAB som förklarades i teoridelen, se sektion 3.4.1. Anledningen till att CEILAB används är för att den projicerade miljön medför att ljusintensitets parametern L är väl lämpad för att detektera vägens linjer samtidigt som b-parametern kan användas för att detektera stopplinjer. Datorseendealgoritmen har en undre och en övre gräns för att filtrera bilder så att endast väg och stopplinjer återstår.

Probabilistisk Hough transform används för att utföra linjedektion samt stopplinjedektion på bilden med filtreerde väg- och stopplinjer. Hough transformen utförs separat för väg- och stopplinjer då linjedektionen har olika satta parametrar och för att skilja på de olika typerna av linjer. Resultatet från den probabilistiska Hough transformen är en vektor med alla detekterade väglinjer och en vektor med alla detekterade stopplinjer. Vägen har endast två väglinjer som ska detekteras så väglinjerna sorteras beroende på om de tillhör höger eller vänster väglinje beroende på linjens lutning. När linjerna sorterats beräknas två medelvärden för lutningarna, ett för vänster och ett för höger väglinje, och de två medelvärdena representerar de slutgiltiga detekterade väglinjerna. Var den vänstra och högra skär en definierad horisontell linje beräknas och är de parametrar som skickas till bilens reglersystem.

Detektion av stopplinjer baserar sig på antalet detekterade linjer. För att undvika falska stopplinjer krävdes att resultatet av stopplinjernas houghtransform detekterade fler linjer än ett satt värde. Gruppen implementerade även en maskning för att undvika detektion av stopplinjer som låg för långt till vänster och höger. Detta gjordes för att undvika



UNTER

falska stopplinjer från filen bilen inte färdades på. Om alla krav var uppfyllda för en stopplinje rapporterades det till bilens reglersystem.

8 SLUTSATSER

Att få en RC-bil att köra autonomt på ett projicerat vägnät var en stor utmaning. Många olika delar var tvungna att samspela, från datorseendet i RPI:n, till styrregleringen till sensormodulen. I följande avsnitt tar dokumentet upp de slutsatser gruppen har dragit utifrån det utförda projektarbetet.

8.1 Slutsats Kommunikation

UART-kommunikationen som användes för att kommunicera mellan moduler ansågs vara ett väl valt bussprotokoll. Den typ av kommunikation som krävde mest dataöverföring över bussen var att skicka styrutslagen från RPI:n till styr-AVR:n. Dessa skickades varje gång styralgoritmen på RPI:n hade räknat ut att ett nytt styrutslag skulle behövas för att följa referensen och då detta kunde ske med ojämna tidsmellanrum var den asynkrona busskommunikationen välanpassad. All annan kommunikation skedde mindre ofta och även då ofta med ojämna tidsmellanrum. Välgenomtänkta placeringar av läsning och skick av data i de olika modulernas kod räckte i de flesta fall för att uppnå en intermodulär kommunikation som uppfyllde projektets krav.

8.2 Slutsats Styrning

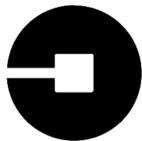
Slutsatsen gruppen drog om styrningen var att den bör anpassas specifikt efter aktuellt vägsegment då det var svårt att hitta en generell reglering för hela vägnätet. Gruppen kom istället att använda tre olika regleringar, en på normal väg och två olika i en rondell beroende på om bilen var påväg in i, befann sig i eller var påväg ut ur rondellen. Denna lösning fungerade mycket bra under utförda tester och det rekommenderas att framtida projekt använder en liknande implementering.

8.3 Slutsats Datorseende

Majoriteten av projektet bestod utav datorseende. Mycket tid gick åt till att få miljön där bilen opererade att fungera så bra som möjligt. Genom olika kalibreringar gick det till sist att få datorseendet att se väglinjerna nere i Visionen. För att detektera väg och kunna följa dess väglinjer krävdes ett robust datorseende. Detta då Visionen till stor del hade störningar i golvet i form av ”blänk” från andra närliggande salar. Genom att undersöka olika bildbehandlingar lyckades till sist ett robust datorseende att implementeras.

8.4 Allmänna Slutsatser

Detta projekt hade stor vikt i området datorseende och datorseendet kom att ta den största delen av projekttiden. Gruppen hade i början av projektet utvecklat sitt datorseende efter miljön uppe i salen MUXEN detta kom att bli en väldigt stor tidsbov då miljön i salen Visionen var något helt annat. Hade projektet gjorts om så hade gruppen mycket tidigare tagit sig till Visionen för att direkt utveckla datorseendet i korrekt miljö .

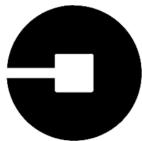


UNTER

Att installera programbibliotek var något gruppen tidigare inte hade gjort. Både programbibliotken till datorseendet, OpenCV, och programbiblioteket till optimeringsalgoritmen, igraph, behövdes installeras. Detta kom att ta längre tid och vara svårare än vad gruppen hade uppskattat. Om projektgruppen vetat detta hade programmbiblioteken i ett tidiga-re stadie installerat dessa bibliotek och rådfrågat en handledare.

Under projektets gång hade projektgruppen en del felkopplingar på virkortet som orsakade oväntat beteende. Projektgruppen kommer i fortsättningen vara extra nogranna för att undvika sådana kopplingar i största möjliga mån för att undvika tidskrävande felsökning och frustration.

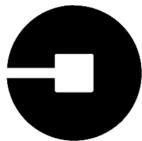
Slutligen vill gruppen nämna att projektet har varit väldigt lärorikt. Projektgruppen har lärt sig mycket om AVR mikrodatorer och hur de används, hur datorseende fungerar och vilka utmaningar som uppstår samt fått en känsla för hur ett projekt utförs i industrin.



UNTER

REFERENSER

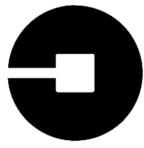
- [1] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Athena Scientific, 1998, 1-886529-02-7.
- [2] R. Szeliski, “Computer vision: Algorithms and applications.”
- [3] Atmel, “LiU, ISY, VanHeden,” <https://docs.isy.liu.se/pub/VanHeden/DataSheets/atmega1284p.pdf>, [Online; accessed may 15, 2019].



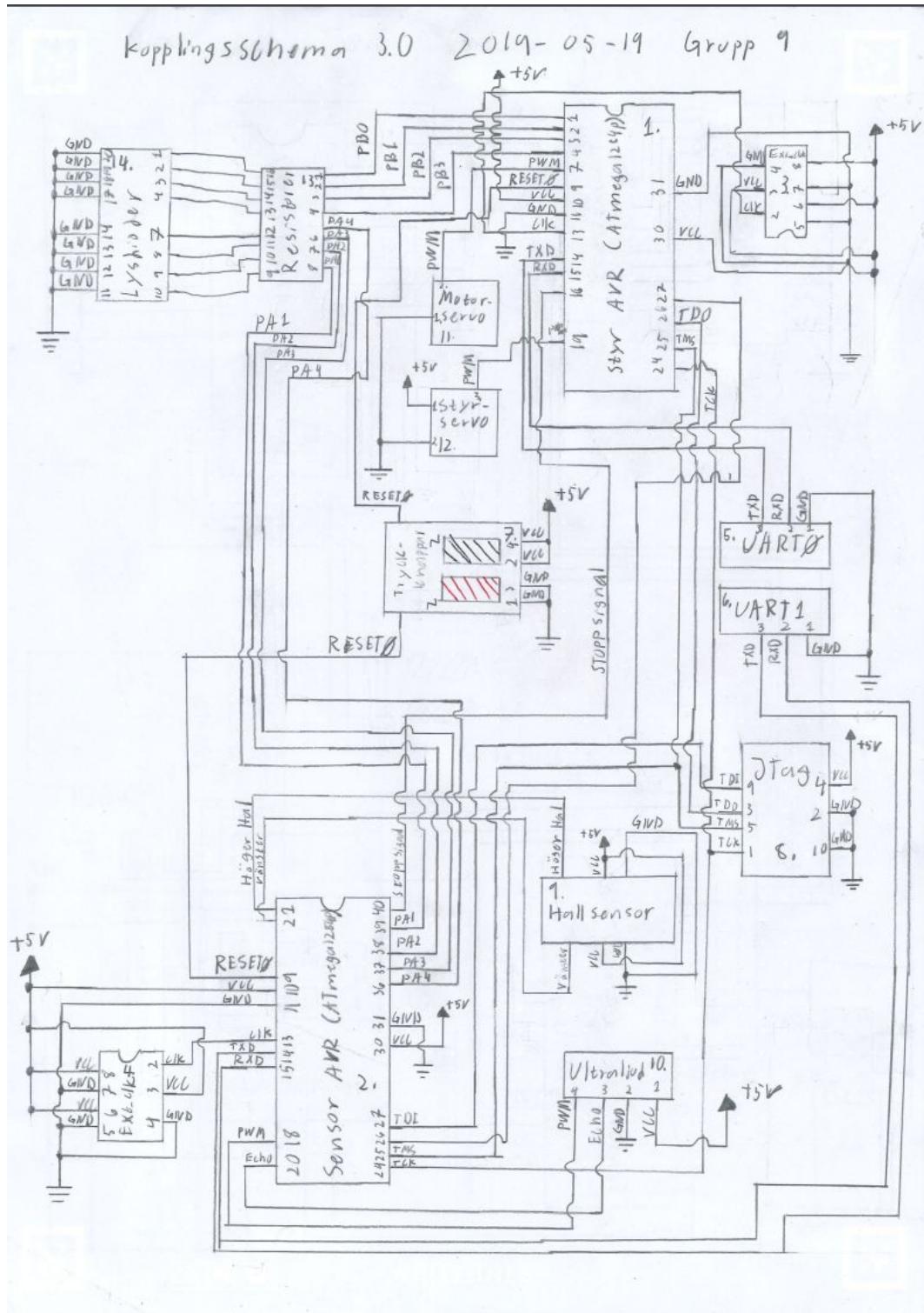
UNTER

- A KRAVSPECIFIKATION
- B BANSPECIFIKATION OCH TÄVLINGSREGLER
- C SYSTEMSKISS
- D PROJEKTPLAN
- E DESIGNSPECIFIKATION
- F FÖRSTUDIE KOMMUNIKATION
- G FÖRSTUDIE STYRNING
- H FÖRSTUDIE SENSOR
- I ANVÄNDARMANUAL
- J KOPPLINGSSCHEMA AUTONOM TAXI

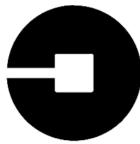
I figur 14 presenteras gruppens kopplingsschema som användes på den färdiga produkten.



UNTER



Figur 14: Kopplingsschema för den autonoma taxibilen.

**UNTER****K PROGRAMLISTNING**

Nedan följer ett urval ur den programkod som användes i projektet.

K.0.1 main.cc

```
#include <iostream>
#include <string>
#include <pthread.h>
#include <semaphore.h>
#include <utility>
#include <vector>
#include <time.h>
#include <igraph.h>
#include <sstream>
#include <iomanip>

#include <ctime>
#include <unistd.h>
#include <fstream>
#include <iostream>
#include <raspicam/raspicam_cv.h>

//Headers from OpenCV
#include <opencv2/core.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

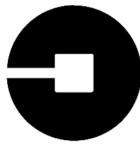
#include "UART.h"
#include "Mode.h"
#include "Manual.h"
#include "Semi_Autonomous.h"
#include "File_Handler.h"

#include "camera/cameracapture.h"
#include "camera/image_transforms.h"
#include "camera/regulation_parameters.h"
#include "Optimization/path_generation.h"

#define BILLION 1E9
#define MILLION 1E6
using namespace std;

// globala variabler
pthread_mutex_t mutex; // initialize mutex
std::vector<double> control_parameters{0,0};
double dt;
bool stop_line_detected = false;
bool stop_line_cooldown = false;
bool in_roundabout = false;
std::vector<double> distances{};

double error_right{};
double error_left{};
```



UNTER

```
double previous_error_right=0;
double previous_error_left=0;
double temp = 0;
bool transition = true;
bool display_references = true;

uint16_t out_right{};
uint16_t out_left{};
uint16_t prev_out_right=0;
uint16_t prev_out_left=0;

// Lter anvndaren mata in startnod och destinationer
void get_dest_from_user(int *start_node, std::vector<int> *dest_vec)
{
    std::cout << "Enter start node: ";
    std::cin >> *start_node;

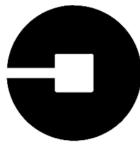
    std::string input_str{};
    std::cout << "Enter destinations: ";
    std::cin.ignore(sizeof(char)); // Remove the newline symbol from the buffer
    std::getline(std::cin, input_str, '\n');
    std::istringstream isstream{input_str};

    int tmp{};
    while( isstream >> tmp )
    {
        dest_vec->push_back(tmp);
    }
}

// Skapar en lista med semiautonoma instruktioner fr en rutt som besker alla destinationsnoder
// med start i startnod som hmtas frn anvndare. (Autonom krning)
void generate_route(vector<Command> * co_vec)
{
    igraph_t g;

    // Skapa ett nodntverk med hjlp av bgar. En bge anges som vanliga parametrar i fljande
    // format: startnod, slutnod.
    igraph_small(&g, 19, IGRAPH_DIRECTED,
                0,2,           1,18,
                2,4,           3,1,
                4,10,          5,10, 5,3,
                6,5,           7,6,
                8,5, 8,14,     9,8,
                10,8, 10,11,   11,12,
                12,13,         13,0, 13,15,
                14,16,         15,16, 15,7,
                16,13, 16,9,   17,15,
                18,17,
                -1);

    // Skapa en vektor som innehller bgarnas vikter. OBS: mste anges i samma ordning som bgarna
    // i igraph_small().
    std::vector<double> weight_vec{31,      // From node 0
                                    30,      // From Node 1
                                    23,      // From Node 2
                                    15,      // From Node 3
```



UNTER

```
6,      // From Node 4
9, 10, // From Node 5
12,    // From Node 6
14,    // From Node 7
15, 32, // From Node 8
6,      // From node 9
8, 12, // From node 10
9,      // From node 11
6,      // From node 12
16, 16, // From node 13
6,      // From node 14
8,18,   // From node 15
8,22,   // From node 16
13,    // From node 17
11};   // From node 18

int start_node{};
std::vector<int> dest_vector();
get_dest_from_user(&start_node, &dest_vector);

generate_full_command_list(&g, co_vec, &weight_vec, start_node, dest_vector); // Skapa den
// fulla kommandolistan med semiautonoma instruktioner

std::cout << "-----COMMAND LIST-----" << std::endl;
for (auto it = co_vec->begin(); it < co_vec->end(); it++)
{
    std::cout << it->get_command_name() << " " << it->get_command_argument() << std::endl;
}
std::cout << std::endl;
igraph_destroy(&g);
}

/* VANLIG VGREGLERING
 * Frsker hitta en hgerlinje att reglera efter. Om hgerlinje
 * inte upptcks s frsker bilen hitta en vnsterlinje att reglera efter.
 */
void control_road(int fd_steer, int fd_sensor)
{
    double Kp_distance_right=12;
    double Kd_distance_right=7;

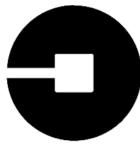
    double Kp_distance_left=12;
    double Kd_distance_left=7;

    double reference_left_distance = -15;
    double reference_right_distance = 340;

    double left_limit_lower = -1000;
    double left_limit_upper = 1000;

    double right_limit_lower = -1000;
    double right_limit_upper = 1000;

    if(distances.back() != 0
        && distances.back() > right_limit_lower
        && distances.back() < right_limit_upper) //Om hger linje finns reglera p hger
    {
        error_right = distances.back() - reference_right_distance;
```



UNTER

```
    if( transition ) // Berkna ej derivata om det      frsta mtningen
{
    temp = 2515 + error_right*Kp_distance_right;
    transition = false;
}
else
{
    temp = 2515 + error_right*Kp_distance_right
        +(error_right-previous_error_right)/dt*Kd_distance_right;
}

out_right = temp;

if((out_right != prev_out_right) && (out_right < 3333)) // Se till att inte skicka ondiga
vrdens med databussen
{
    write(fd_steer,&out_right,2);
    tcdrain(fd_steer);
}
previous_error_right = error_right;
prev_out_right = out_right;

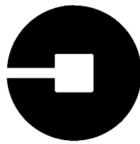
}
else if (distances.front() != 0
&& distances.front() > left_limit_lower
&& distances.front() < left_limit_upper){ // Om vnster linje finns, reglera p vnster.
error_left = distances.front() - reference_left_distance;

if (transition) // Berkna ej derivata om det      frsta mtningen .
{
temp = 2515 + error_left*Kp_distance_left;
transition = false;
}
else
{
temp = 2515 + error_left*Kp_distance_left
+(error_left-previous_error_left)/dt*Kd_distance_left;
}

out_left = temp;

if((out_left != prev_out_left) && (out_left > 1800)) // Filtrera bort ondiga utsignaler (fr sm eller samma som innan)
{
write(fd_steer,&out_left,2); // send out servo output
tcdrain(fd_steer);
}
previous_error_left = error_left;
prev_out_left = out_left;
}

/*
 * REGLERING INUTI RONDELL
 * Anvnds fr att flja rondellens innerkant. Aktiveras inuti rondellen.
 * (om inte bilen ska ta den frsta utfarten)
*/
```



UNTER

```
* PRIO: 1. Vnsterlinje
*         2. Hgerlinje
*/
void control_roundabout(int fd_steer, int fd_sensor)
{
    double Kp_distance_right=15;
    double Kd_distance_right=15;

    double Kp_distance_left=15;
    double Kd_distance_left=15;

    double Ki_distance_left=0;
    double Ki_distance_right=0;

    double reference_left_distance = 30; // 30
    double reference_right_distance = 420; //420

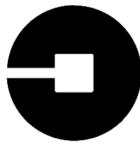
    double left_limit_lower = -1000; // (-30 mvh emil)
    double left_limit_upper = 1000;

    double right_limit_lower = -1000;
    double right_limit_upper = 1000;

    // Vnsterreglering
    if (distances.front() != 0
        && distances.front() > left_limit_lower
        && distances.front() < left_limit_upper)
    {
        error_left = distances.front() - reference_left_distance;

        if (transition) // Berkna ej derivata om det      frsta mtningen
        {
            temp = 2515 + error_left*Kp_distance_left;
            transition = false;
        }
        else
        {
            temp = 2515 + error_left*Kp_distance_left
                +(error_left-previous_error_left)/dt*Kd_distance_left;
        }
        out_left = temp;

        if(out_left != prev_out_left) // Anvnd inte bussen om det      samma vrde som innan
        {
            write(fd_steer,&out_left,2);
            tcdrain(fd_steer);
        }
        previous_error_left = error_left;
        prev_out_left = out_left;
    }
    else if(distances.back() != 0
        && distances.back() > right_limit_lower
        && distances.back() < right_limit_upper) //Om hger linje finns reglera p hger (filtrera
                                                //bort orimliga vrden)
    {
        error_right = distances.back() - reference_right_distance;
```



UNTER

```
    if(transition)
{
    temp = 2515 + error_right*Kp_distance_right;
    transition = false;
}
else
{
    temp = 2515 + error_right*Kp_distance_right
        +(error_right-previous_error_right)/dt*Kd_distance_right;
}

    out_right = temp;

    if((out_right != prev_out_right)) // Anvnd inte bussen om det      samma vrde som tidigare
{
    write(fd_steer,&out_right,2); // Skicka vrdet p bussen
    tcdrain(fd_steer);
}
    previous_error_right = error_right;
    prev_out_right = out_right;

}
}

/* REGLERING IN TILL RONDELLEN
 * Anvnds fr att ta sig in i rondellen frn vanlig vg. aktiveras efter
 * den bl linjen som      placerad precis innan sjlva rondellen.
 * PRIO: 1. Hgerlinje
 *       2. Vnsterlinje
 */
void control_entry(int fd_steer, int fd_sensor)
{
    double Kp_distance_right=21;
    double Kd_distance_right=13;

    double Kp_distance_left=10;
    double Kd_distance_left=3;

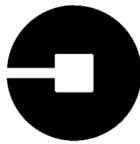
    double reference_left_distance = -50;
    double reference_right_distance = 320;

    double left_limit_lower = -1000;
    double left_limit_upper = 1000;

    double right_limit_lower = -1000;
    double right_limit_upper = 1000;

    if(distances.back() != 0
        && distances.back() > right_limit_lower
        && distances.back() < right_limit_upper) //Om hger linje finns reglera p hger
    {
        error_right = distances.back() - reference_right_distance;

        if(transition) // Berkna inte derivatan om det      frsta mtningen
    {
```



UNTER

```
temp = 2515 + error_right*Kp_distance_right;
transition = false;
}
else
{
    temp = 2515 + error_right*Kp_distance_right
        +(error_right-previous_error_right)/dt*Kd_distance_right;
}

out_right = temp;

if((out_right != prev_out_right) && (out_right < 3333))
{
    write(fd_steer,&out_right,2);
    tcdrain(fd_steer);
}
previous_error_right = error_right;
prev_out_right = out_right;

}

else if (distances.front() != 0
&& distances.front() > left_limit_lower
&& distances.front() < left_limit_upper){ // vnster
error_left = distances.front() - reference_left_distance; // kolla tecken

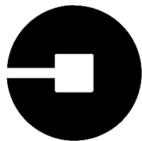
if(transition)
{
temp = 2515 + error_left*Kp_distance_left;
transition = false;
}
else
{
temp = 2515 + error_left*Kp_distance_left
+(error_left-previous_error_left)/dt*Kd_distance_left;
}

out_left = temp;

if((out_left != prev_out_left) && (out_left > 1800))
{
    write(fd_steer,&out_left,2); // send out servo output
    tcdrain(fd_steer);
}
previous_error_left = error_left;
prev_out_left = out_left;
}
}

void enter_roundabout(int fd_steer, int fd_sensor)
{
in_roundabout = true;

while(1)
{
    pthread_mutex_lock(&mutex);
    control_entry(fd_steer, fd_sensor); // Utför ingångsregleringen för rondell
    pthread_mutex_unlock(&mutex);
}
```



UNTER

```
    if(!stop_line_cooldown)
{
    if(stop_line_detected){ // Om bllinje upptcks, byt till ny reglering
        stop_line_cooldown = true; // Frhindra dubbel detektion av samma linje.
        transition = true;
        break;
    }
} else
{
    if(!stop_line_detected) // Tillt nya detektioner nr den gamla linjen borta
        stop_line_cooldown = false;
}
}

void exit_roundabout(int fd_steer, int fd_sensor)
{
    char ask = 1; // Anvnds fr att begra krstrcka frn sensormodulen.
    write(fd_sensor, &ask, 1);
    tcdrain(fd_sensor);

    double start_point = read_UART16(fd_sensor)/1000; // Ls & spara krstrcka

    write(fd_sensor, &ask, 1);
    tcdrain(fd_sensor);

    double current_point = read_UART16(fd_sensor)/1000;

    while((current_point - start_point) < 1.7)
    {
        pthread_mutex_lock(&mutex);
        control_entry(fd_steer, fd_sensor);
        pthread_mutex_unlock(&mutex);

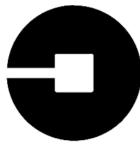
        write(fd_sensor, &ask, 1);
        tcdrain(fd_sensor);

        current_point = read_UART16(fd_sensor)/1000;
    }

    in_roundabout = false;
    transition = true; // Tell next control loop to not use the previous error (would give crazy
    result)
}

//Bilens krprogram.
void * communication_thread(void *unused)
{
    cout << "var vnlig och vlj lge" << endl;

    // open ports in read/write, not as controlling terminal and
    // in sync mode (OS tries to guarantee that data is on disk before reading)
    // Open file descriptor for steering AVR
```



UNTER

```
int fd0 = open("/dev/ttyUSB0", O_RDWR | O_NOCTTY | O_SYNC);

if(fd0 < 0)
    cout << "Port0 was not opened" << endl;

// Open file descriptor for sensor AVR
int fd1 = open("/dev/ttyUSB1", O_RDWR | O_NOCTTY | O_SYNC);

if(fd1 <0)
    cout << "Port1 was not opened" << endl;

// konfigurera UART-kommunikationen
UART_config(fd0, B9600, 0); // konfiguration av UART0
UART_config(fd1, B9600, 0); // konfiguration av UART1

string mode_select;
cin >> mode_select; // kanonisk inläsning av k r l get

if(mode_select == "m"){ // manual
    cout << "enter manual mode" << endl;
    char activate_man = 1;
    write(fd0, &activate_man, 1); // activate manual mode in steer AVR
    tcdrain(fd0);

    write(fd1, &activate_man, 1); // activate manual mode in sensor AVR
    tcdrain(fd1);

    Manual manual{fd0, fd1};
    manual.read_steering();
} else
{
    vector <Command> command_list{};

    if (mode_select == "s"){ // semi-autonomous. Användaren skapar kommandolista.
        cout << "enter semi-autonomous mode" << endl;
        char activate_SA = 2;
        write(fd0, &activate_SA, 1); // activate semi-auto mode in steer AVR
        tcdrain(fd0);

        write(fd1, &activate_SA, 1); // activate semi-auto mode in sensor AVR
        tcdrain(fd1);

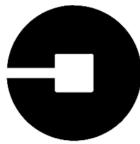
        Semi_Autonomous semi_auto {0, 0};
        command_list = semi_auto.read_command();

    } else if (mode_select == "a"){ // Autonomt k r l ge ! Skapar kommandolista utifr n noddata frn användaren.
        cout << "enter autonomous mode" << endl;
        char activate_A = 3;
        write(fd0, &activate_A, 1); // activate auto mode in steer AVR
        tcdrain(fd0);

        write(fd1, &activate_A, 1); // activate auto mode in sensor AVR
        tcdrain(fd1);

        int start_node{};
        vector <int> dest_vec{};

    }
}
```



UNTER

```
generate_route(&command_list);
}

// Exekvering av kommandolista.
stop_line_cooldown = true; // Ignorera stopplinje vid start.
while(!command_list.empty()) // Ls igenom kommandolistan
{
    transition = true;
    uint16_t output = command_list.front().get_command_code();
    write(fd0, &output, 2);
    tcdrain(fd0);

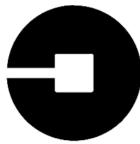
    if(command_list.front().get_command_name() == "forward")
    {
        cout << "k r framt" << endl;

        int counter = 0;
        while(command_list.front().get_command_argument() > counter)
        {
            if(!stop_line_cooldown)
            {
                if(stop_line_detected)
                {
                    counter++;
                    stop_line_cooldown = true;
                    cout << "jag sg en bl linje" << endl;
                }
            }
            else
            {
                if(!stop_line_detected)
                stop_line_cooldown = false;
            }
        }
        pthread_mutex_lock(&mutex);
        control_road(fd0, fd1);
        pthread_mutex_unlock(&mutex);
    }
    else if (command_list.front().get_command_name() == "roundabout")
    {
        cout << "k r i rondell" << endl;
        if(command_list.front().get_command_argument() == 1)
        {
            stop_line_cooldown = true; // slutar reglera d en stoppline har hittats
            transition = true;
            enter_roundabout(fd0, fd1);

            cout << "byter till exit" << endl;
            transition = true;
            exit_roundabout(fd0, fd1); // slutar reglera d en viss strcka har krts

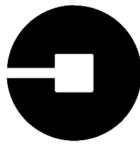
            cout << "avslutar exit" << endl;

            while(!stop_line_detected) // avslutar d en bl linje hittas
        }
    }
}
```



UNTER

```
{  
    pthread_mutex_lock(&mutex);  
    control_road(fd0, fd1);  
    pthread_mutex_unlock(&mutex);  
}  
    transition = true;  
} else  
{  
    int counter = 1;  
  
    stop_line_cooldown = true;  
    transition = true;  
    enter_roundabout(fd0, fd1); // Reglera in i rondell.  
  
    cout << "inne i rondell" << endl;  
  
    while(command_list.front().get_command_argument() > counter) // s lnge vi inte ser en  
stopplinje och ingen cooldown  
    {  
        if(!stop_line_cooldown)  
        {  
            if(stop_line_detected)  
            {  
                counter++;  
                stop_line_cooldown = true;  
            }  
  
        } else  
        {  
            if(!stop_line_detected)  
                stop_line_cooldown = false;  
        }  
        pthread_mutex_lock(&mutex);  
        control_roundabout(fd0, fd1);  
        pthread_mutex_unlock(&mutex);  
    }  
  
    cout << "byter till exit" << endl;  
    transition = true;  
    exit_roundabout(fd0, fd1);  
  
    while(!stop_line_detected)  
    {  
        pthread_mutex_lock(&mutex);  
        control_road(fd0, fd1);  
        pthread_mutex_unlock(&mutex);  
    }  
    //stop_line_cooldown = true;  
}  
  
}  
else if (command_list.front().get_command_name() == "stop")  
{  
    cout << "stannar" << endl;  
    sleep(2);  
    transition = true;  
}
```



UNTER

```
output = 75; // skicka ok-signal till styr AVR
write(fd0, &output, 2);
tcdrain(fd0);

cout << "byter kommando" << endl;
command_list.erase(command_list.begin());
}
}

close(fd0);
close(fd1);
cout << "end of line" << endl;
}

///////////////////////////////
// Camera_thread //
///////////////////////////////

void *camera_thread(void *unused/*int argc, char** argv*/)
{
    //Defining used varaibles
    cv::Mat frame; //Stores image captured from camera
    cv::Mat src;   //Stores the undistorted iamge
    cv::Mat dst;   //Stores the processed image

    //Stores maps for undistorting the cameralens fisheye effect
    std::pair<cv::Mat, cv::Mat> undistortion_maps;

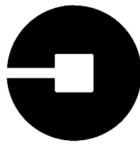
    std::pair<std::vector<cv::Vec4i>, std::vector<cv::Vec4i>> lines;           //Stores lines extracted
    from image
    std::vector<cv::Vec4i> left_lines; //Stores detected lines with positive incline
    std::vector<cv::Vec4i> right_lines; //Stores detected lines with negative incline
    std::vector<cv::Vec4i> road_lines;
    std::vector<cv::Vec4i> stop_lines;
    std::vector<cv::Vec4i> sorted_stop_lines;

    //bool stop_line_detected;
    std::pair<double, double> detected_stop_line;
    int stop_counter = 2;

    //y = k*x + m
    //Stores the average values for k and m of detected left lines
    std::pair<double, double> left_line_parameters;
    //Stores the average values for k and m of detected right lines
    std::pair<double, double> right_line_parameters;
    //stores distance from line intersect with bottom of image and center
    double distance_center_left = 0;
    double distance_center_right = 0;

    //Varaibles used for time measurement
    double e1 = 0;
    double e2 = 0;
    double time = 0;

    raspicam::RaspiCam_Cv Camera;
```



UNTER

```
//set camera params
Camera.set(CV_CAP_PROP_FORMAT, CV_8UC3);
Camera.set(CV_CAP_PROP_FRAME_WIDTH, 640/2);
Camera.set(CV_CAP_PROP_FRAME_HEIGHT, 480/2);
Camera.set(CV_CAP_PROP_EXPOSURE, 12); // shutter
//Open camera
std::cout<<"Opening Camera..."<<std::endl;
if (!Camera.open()){
    std::cerr<<"Error opening the camera"<<std::endl;
    //return void;
}
//Start capture
if(!Camera.grab()){
    std::cerr<<"grab failed"<<std::endl;
}
Camera.retrieve (frame);

//undistortion_maps = generate_undistortion_maps(frame);

//Run the program until user presses ESC key
while(1)
{
    Camera.grab();
    Camera.retrieve (frame);
    //cv::imshow("raw",frame);

    // If the frame is empty, break
    if (frame.empty())
break;

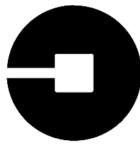
    //Apply distortion parameters to captured frame to remove fisheye effect
    // undistort(frame, src, undistortion_maps);
    frame.copyTo(src);

    //Apply image transforms and perform probabilistic hough transform to extract lines
    lines = extract_lines(src, dst, in_roundabout); // image_transformation.cpp
    road_lines.clear();
    stop_lines.clear();
    road_lines = std::get<0>(lines);
    stop_lines = std::get<1>(lines);

    //Sort the lines
    left_lines.clear();
    right_lines.clear();
    sorted_stop_lines.clear();

    sort_lines(left_lines, right_lines, road_lines);

    /* if(!in_roundabout)
    {
        sort_stop_lines(stop_lines, sorted_stop_lines);
        if(sorted_stop_lines.size() >= 1)
        {
            if(stop_counter <= 0)
            {
                stop_line_detected = true;
            }
        }
    }
}
```



UNTER

```
else
{
    stop_counter = stop_counter - 1;
    stop_line_detected = false;
}
}
else
{
    stop_counter = 3;
    stop_line_detected = false;
}
} */
//else if(stop_lines.size() >= 5)
if(stop_lines.size() >= 1)
{
    if(stop_counter <= 0)
    {
        stop_line_detected = true;
    }
    else
    {
        stop_counter = stop_counter - 1;
        stop_line_detected = false;
    }
}
else
{
    stop_counter = 2;
    stop_line_detected = false;
}

//cout << stop_line_detected << endl;
//Process the left lines
left_line_parameters = average(left_lines);
draw_average_line(dst, left_line_parameters);

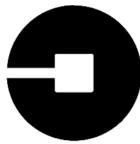
//Processing the right lines
right_line_parameters = average(right_lines);
draw_average_line(dst, right_line_parameters);

//Calculate regulation parameters
pthread_mutex_lock(&mutex);
e2 = cv::getTickCount();

dt = (e2 - e1)/cv::getTickFrequency();

/* #####TESTREGLERING AVSTND#####
 * TESTREGLERING AVSTND
 *
 * Avstndet i x-led frn d den vnsta respektive hgra linjen till
 * mitten av bilden liker i en publik vektor kallad "distances".
 * Avstdent fr vnster linje det frsta elementet och avstndet
 * fr hger linje det andra.
 *
 * #####TESTREGLERING AVSTND#####
 */

//does the calculation (x=(y-m)/k) and subtracts x-value for center of src
```



UNTER

```
distances.clear();
if(std::get<0>(left_line_parameters) == 0) //undvikar division med 0 om vnsterlinje saknas
{
    distances.push_back(0);
}
else
{
    distance_center_left = (src.rows*0.85 - std::get<1>(left_line_parameters)) / std::get<0>(left_line_parameters);
    distances.push_back(distance_center_left);
}
if(std::get<0>(right_line_parameters) == 0) //undvikar division med 0 om hgerlinje saknas
{
    distances.push_back(0);
}
else
{
    distance_center_right = (src.rows*0.85 - std::get<1>(right_line_parameters)) / std::get<0>(right_line_parameters);
    distances.push_back(distance_center_right);
}

//std::cout << "hger " << distances.back() << std::endl;
//std::cout << "vnster " << distances.front() << std::endl;

e1 = cv::getTickCount();
pthread_mutex_unlock(&mutex);

//Show results
cv::imshow("Detected Lines", dst);

//Wait and Exit. Press ESC or Space on keyboard to exit
char c=(char)cv::waitKey(25);
if(c==0x1B) //esc och space in hex
{
    break;
}

}

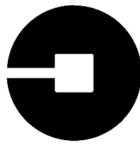
// When everything done, release the video capture object
Camera.release();

// Closes all the frames
cv::destroyAllWindows();
return NULL;
}

// threaded communication and computer vision
int main()
{
pthread_mutex_init(&mutex, NULL);

pthread_t communication_thread_handle;
pthread_t camera_thread_handle;

pthread_create(&communication_thread_handle, NULL, communication_thread, 0);
pthread_create(&camera_thread_handle, NULL, camera_thread, 0);
```



UNTER

```
pthread_join(communication_thread_handle,NULL);
pthread_join(camera_thread_handle,NULL);
return 0;
}
```

K.0.2 image_transforms.cpp

```
/*#####
 * This file contains the implementation of functions needed to detect
 * lines in an image.
 * Programmerare:Dennis sterdahl
 * DATUM: 2019-05-21
 *#####
 */

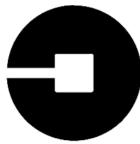
//Including headers

//std headers
#include <utility>
#include <iostream>
#include <vector>

//opencv headers
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
//own header
#include "image_transforms.h"

//This function detects the lines in an image using probabilistic hough
//transform. The result is detected roadlines and detected stoplines
std::pair<std::vector<cv::Vec4i>, std::vector<cv::Vec4i>> extract_lines(cv::Mat& src, cv::Mat&
dst, bool in_roundabout)
{
    if(src.empty())//if given image is empty, abort.
    {
        std::pair<std::vector<cv::Vec4i>, std::vector<cv::Vec4i>> foo{};
        return foo;
    }
    //cv::imshow("RAW", src);
    cv::Mat blueMat, greyMat, labMat;
    bool stop_line_detected = false;

    //Mask
    // mask of top of frame
    if(in_roundabout) // inuti rondell
    {
        cv::Point road_points[1][5];
        const cv::Point* ppt[1]= {road_points[0]};
        int npt[] = {5};
        road_points[0][0] = cv::Point(0, 0);
        road_points[0][1] = cv::Point(0, src.rows*0.30);
        road_points[0][2] = cv::Point(src.cols*0.5, src.rows*0.13);
```



UNTER

```
road_points[0][3] = cv::Point(src.cols, src.rows*0.30);
road_points[0][4] = cv::Point(src.cols, 0);
cv::fillPoly(src, ppt, npt, 1, cv::Scalar(255, 255, 255), cv::LINE_4);
}else// utanför rondell
{
    cv::Point road_points[1][5];
    const cv::Point* ppt[1]= {road_points[0]};
    int npt[] = {5};
    road_points[0][0] = cv::Point(0, 0);
    road_points[0][1] = cv::Point(0, src.rows*0.38);
    road_points[0][2] = cv::Point(src.cols*0.5, src.rows*0.18);
    road_points[0][3] = cv::Point(src.cols, src.rows*0.38);
    road_points[0][4] = cv::Point(src.cols, 0);
    cv::fillPoly(src, ppt, npt, 1, cv::Scalar(255, 255, 255), cv::LINE_4);
}

// mask of bottom of frame
cv::rectangle(src, cv::Point(0,src.rows), cv::Point(src.cols, 0.85*src.rows),
    cv::Scalar(255, 255, 255), cv::FILLED);

//cv::imshow("efter nedre maskning",src);
cv::cvtColor(src, labMat, cv::COLOR_BGR2Lab);
cv::cvtColor(src, greyMat, cv::COLOR_BGR2GRAY);

//detect blue line
cv::Mat labMatTest;
cv::cvtColor(src, labMatTest, cv::COLOR_BGR2Lab);
cv::Scalar lower_blue_limit = cv::Scalar(0, 0, 0); //cv::Scalar(90, 75, 75);
cv::Scalar upper_blue_limit = cv::Scalar(239, 157, 124); //(110, 90, 90)
cv::inRange(labMatTest, lower_blue_limit, upper_blue_limit, blueMat);

//Greyscale image
cv::cvtColor(src, greyMat, cv::COLOR_BGR2GRAY);

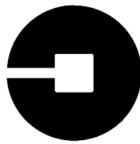
// Threshold
cv::threshold(greyMat, edgeMat, 160, 255, cv::THRESH_BINARY_INV);

cv::Mat dilated_blueMat;
cv::Mat stop_line_kernel = cv::getStructuringElement(cv::MORPH_RECT,
    cv::Size(1, 9));
cv::dilate(blueMat, dilated_blueMat, stop_line_kernel);

cv::subtract(edgeMat, dilated_blueMat, edgeMat);
cv::rectangle(blueMat, cv::Point(0, 0), cv::Point(0.25*blueMat.cols, blueMat.rows),
    cv::Scalar(0, 0, 0), cv::FILLED);

//Copy edges to the images that will display the results in BGR
cv::cvtColor(edgeMat, dst, cv::COLOR_GRAY2BGR);

//Detect and draw detected road lines
std::vector<cv::Vec4i> lines;
```



UNTER

```
cv::HoughLinesP(edgeMat, lines, 1, CV_PI/180, 50, 35, 40);

for(std::size_t i = 0; i < lines.size(); i++)
{
    cv::Vec4i l = lines[i];
    cv::line(dst, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]), cv::Scalar(0, 0, 255), 3, cv::LINE_AA);
}

//Detect and draw detected blue lines
std::vector<cv::Vec4i>blue_lines;
cv::HoughLinesP(blueMat, blue_lines, 1, CV_PI/180, 60, 50, 10);
for(std::size_t i = 0; i < blue_lines.size(); i++)
{
    cv::Vec4i l = blue_lines[i];
    cv::line(dst, cv::Point(l[0], l[1]), cv::Point(l[2], l[3]), cv::Scalar(155, 155, 0), 3, cv::LINE_AA);
}

//Construct the pair that should be returned
std::pair<std::vector<cv::Vec4i>, std::vector<cv::Vec4i>> road_lines_and_stop_lines = std::make_pair(lines, blue_lines);

return road_lines_and_stop_lines;
}
```

K.0.3 *UART.cc*

```
/*
 * UART.cc
 * This file handles UART communication.
 * Programmer: Nicholas Sepp
 * Date: 2019-04-21
 */

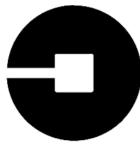
#include "UART.h"

using namespace std;

int UART_config (int fd, int speed, int parity)
{
    struct termios tty;
    memset (&tty, 0, sizeof tty);
    if (tcgetattr (fd, &tty) != 0)
    {
        cout << "error from tcgetattr" << endl;
        return -1;
    }

    cfsetospeed (&tty, speed);
    cfsetispeed (&tty, speed);

    tty.c_cflag &= ~PARENB; // no parity bit
    tty.c_cflag &= ~CSTOPB; // 1 stop bit
    tty.c_cflag &= ~CSIZE; // mask the character size bits
    tty.c_cflag |= CS8; // 8 data bits
```



UNTER

```
tty.c_cflag &= ~CRTSCTS; // no hardware control (do not use handshake)

tty.c_cflag |= (CLOCAL | CREAD); // ignore modem controls,

// turn off canonical mode, use raw input: input characters are passed
// through exactly as they are received, when they are received.
tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL | IXON);
tty.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG | IEXTEN);
//tty.c_lflag = 0;
tty.c_oflag &= ~OPOST;

// fetch bytes as they become available
tty.c_cc[VMIN] = 1;
tty.c_cc[VTIME] = 1;

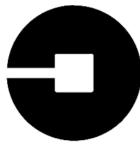
if (tcsetattr (fd, TCSANOW, &tty) != 0)
{
    cout << "error from tcsetattr" << endl;
    return -1;
}
return 0;
}

/* Reads a 16 bit word from file descriptor,
   low byte is sent first and high byte is sent afterwards,
   returns read word as a double */
double read_UART16(int fd)
{
    char high_byte;
    char low_byte;
    uint16_t temp;
    double output;

    while(read(fd, &low_byte, 1) < 0){
        // read the low byte of the 16 bit word
    }
    while(read(fd, &high_byte, 1) < 0){
        // read the high byte of the 16 bit word
    }
    temp = (high_byte << 8) | low_byte; // combine both bytes into word
    output = temp; // convert 16 bit word into double
    //output = output / 100; // divide with 100 because of scaling
    return output;
}

/* Reads total distance from sensor AVR */
double read_total_distance(int fd_sensor)
{
    return (read_UART16(fd_sensor)/100); // divide with 100 because of scaling
}

/* Reads engine output as ticks from steering AVR */
double read_engine(int fd_steer)
{
    return read_UART16(fd_steer);
}
```



UNTER

```
/* Reads servo output as ticks from steering AVR */
double read_servo(int fd_steer)
{
    return read_UART16(fd_steer);
}
```

K.0.4 path_generation.h

```
/*
 * This program creates a graph and generates a path that visits some specified nodes (like a
 * taxi).
 * The shortest paths for each sub-trip is calculated with dijkstras shortest path algorithm.
 * This network problem
 * is classified as NP-Hard. The result is thus an approximation of the optimal solution.
 * Programmer: Emil M rtensson
 * Date: 2019-05-14
 */

#ifndef PATH_GENERATION_H
#define PATH_GENERATION_H

extern "C" // This scope defines C functions for use in C++
{
#include <iigraph.h>

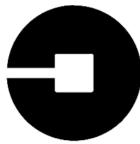
int print_matrix(const igraph_matrix_t *m) {
    long int nrow=igraph_matrix_nrow(m);
    long int ncol=igraph_matrix_ncol(m);
    long int i, j;
    for (i=0; i<nrow; i++) {
        printf("%li:", i);
        for (j=0; j<ncol; j++) {
            printf(" %3.0F", MATRIX(*m, i, j));
        }
        printf("\n");
    }
    return 0;
}
}

/////////// C++ part starts here //////////

#include <iostream>
#include <vector>
#include <array>
#include <cmath>
#include <algorithm>
#include "../Mode.h"

// Enumerators: Road type

enum Road
{
    Forward,
    Roundabout,
    Entrance
}
```



UNTER

```
};

//Print a vector of the igraph_vector_t type
void print_vector(const igraph_vector_t *v)
{
    std::cout << std::endl;
    long int i;
    for (i=0; i<igraph_vector_size(v); i++)
    {
        std::cout << "Node " << i << ":" << VECTOR(*v)[i] << std::endl;
    }
    std::cout << std::endl;
}

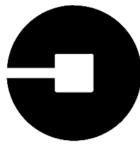
//Write graph to a file (const char *filename is a string)
void graph_to_file(igraph_t *graph, const char *filename)
{
    FILE *output_file;
    output_file = fopen(filename, "w");
    igraph_write_graph_lgl(graph, output_file, 0, 0, false);
    fclose(output_file);
}

//Initialize a graph from file. const char *filename is a regular string
//The graph object must be unitialized.
void file_to_graph(igraph_t *graph, const char *filename)
{
    FILE *input_file;
    input_file = fopen(filename, "r");
    if ( !input_file )
    {
        std::cout << "Missing lgl-file, (.lgl is the format that holds information about the graph)"
        << std::endl;
        return;
    }
    else
    {
        igraph_read_graph_lgl(graph, input_file, 0, IGRAPH_ADD_WEIGHTS_NO, IGRAPH_DIRECTED);
    }
}

// Check if a vertex (node) is inside a roundabout. Returns true if it is.
bool roundabout_slow_check(const igraph_integer_t vertex, const igraph_vector_t *degree_vector)
{
    return VECTOR(*degree_vector)[vertex] > 1;
}

//Check if a vertex (node) is an entrance to a roundabout.
bool entrance_slow_check(igraph_t *graph, const igraph_integer_t vertex, const igraph_vector_t *
    degree_vector)
{
    if( roundabout_slow_check(vertex, degree_vector) )
    {
        return false;
    }

    igraph_vector_t neighbor_vec;
```



UNTER

```
igraph_vector_init(&neighbor_vec, 0);
igraph_neighbors(graph, &neighbor_vec, vertex, IGRAPH_OUT);

// Check if any neighbour is inside a roundabout. If it is, we are looking at a roundabout
// start node.
for(int i = 0; i<igraph_vector_size(&neighbor_vec); i++) // If any neighbour is inside a
    roundabout, return true
{
    if( roundabout_slow_check( VECTOR(neighbor_vec)[i], degree_vector) )
    {
        igraph_vector_destroy(&neighbor_vec);
        return true;
    }
}

//Destroy vectors
igraph_vector_destroy(&neighbor_vec);
return false;
}

// Check every node in the network for roundabout entrances. The result is a vector of bools
// where the index of the element represents the node index.
std::vector<bool> create_entrance_vector(igraph_t *graph, const igraph_vector_t *degree_vector)
{
    std::vector<bool> result;

    for (igraph_integer_t i{}; i < igraph_vector_size(degree_vector); i++)
    {
        result.push_back(entrance_slow_check(graph, i, degree_vector));
    }

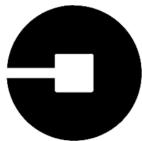
    return result;
}

// Check every node in the network for roundabouts. The result is a vector of bools where the
// index of the element represents the node index.
std::vector<bool> create_roundabout_vector(const igraph_vector_t *degree_vector)
{
    std::vector<bool> result;

    for (igraph_integer_t i{}; i < igraph_vector_size(degree_vector); i++)
    {
        result.push_back(roundabout_slow_check(i, degree_vector));
    }

    return result;
}

// Find the path to the closest destination
// Input:
//     start_node: (pointer to) the node from which the shortest path should start.
//     destination_nodes: (pointer to) a vector of nodes in which the path has to end.
//
// Output:
//     A vector of igraph_integer_t (long integers) that represents each step in the path. The
//     first node is the start node, the last is the closest destination node.
std::vector<igraph_integer_t> find_closest_destination(const igraph_t *graph,
```



```
        std::vector<double> *weight_vec,
        const int start_node,
        const std::vector<int> *destination_nodes)
{
    igraph_vector_t weights; //Create a igraph vector
    igraph_vector_init_copy(&weights, weight_vec->data(), weight_vec->size()); //Copy weight vector

    //Converting the destinations into an igraph_vector_t type. This is required by the dijkstra
    //function below
    std::vector<igraph_real_t> destination_nodes_double(destination_nodes->begin(),
                                                       destination_nodes->end());
    igraph_vector_t igraph_dest_nodes;
    igraph_vector_init_copy(&igraph_dest_nodes,
                           destination_nodes_double.data(),
                           destination_nodes->size());

    //The matrix will store result from dijkstra algorithm
    igraph_matrix_t distance_matrix;
    igraph_matrix_init(&distance_matrix, 1, destination_nodes->size()); // Initialize a matrix with
    // 1 row

    //Dijkstra algorithm. This function call is very long, the arguments are listed on each row
    // separately.
    igraph_shortest_paths_dijkstra(graph,
                                    &distance_matrix,
                                    igraph_vss_1(start_node),
                                    igraph_vss_vector(&igraph_dest_nodes),
                                    &weights,
                                    IGRAPH_OUT);

    //Generate a distance vector from the result
    igraph_vector_t distance_vector;
    igraph_vector_init(&distance_vector, 0);
    igraph_matrix_get_row(&distance_matrix, &distance_vector, 0); //Transfer the first row in
    //matrix into the vector

    igraph_integer_t cheapest_index;
    cheapest_index = igraph_vector_which_min(&distance_vector);

    int closest_node{(*destination_nodes)[cheapest_index]}; // This accesses an element in
    // destination_nodes vector

    std::cout << "The closest node is node " << closest_node << '.' << std::endl;

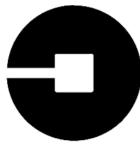
    //Finding the closest path (as a vector of nodes)
    igraph_vector_t path_result;
    igraph_vector_init(&path_result, 0);

    igraph_get_shortest_path_dijkstra(graph, &path_result, nullptr, start_node, closest_node, &
    weights, IGRAPH_OUT);

    // Transfer data from igraph_vector_t to a vector of integers. (For convenience of c++ users)
    std::vector<igraph_integer_t> result;

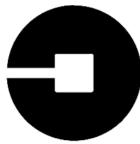
    for (long int i{}; i < igraph_vector_size(&path_result); i++)

```



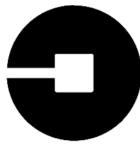
UNTER

```
{  
    result.push_back(VECTOR(path_result)[i]);  
}  
  
// Deallocate resources to prevent memory leaks  
igraph_vector_destroy(&distance_vector);  
igraph_matrix_destroy(&distance_matrix);  
igraph_vector_destroy(&weights);  
igraph_vector_destroy(&path_result);  
  
return result;  
}  
  
// Print a vector  
void print(std::vector<int> const *vec)  
{  
    for (auto it = vec->begin(); it < vec->end(); it++)  
    {  
        std::cout << " " << *it;  
    }  
    std::cout << std::endl;  
}  
  
// COMMAND GENERATION FUNCTIONS  
void add_stop_command(std::vector<Command> *cmd_list)  
{  
    Command *cmd_ptr = new Command("stop", 0);  
    cmd_list->push_back(*cmd_ptr);  
}  
  
void add_forward_command(std::vector<Command> *cmd_list, const int argument)  
{  
    Command *cmd_ptr = new Command("forward", argument);  
    cmd_list->push_back(*cmd_ptr);  
}  
  
void add_roundabout_command(std::vector<Command> *cmd_list, const int argument)  
{  
    Command *cmd_ptr = new Command("roundabout", argument);  
    cmd_list->push_back(*cmd_ptr);  
}  
  
void generate_command_list(std::vector<Command> *cmd_list,  
                          const std::vector<int> *path_vec,  
                          const std::vector<bool> *entrance_vec,  
                          const std::vector<bool> *roundabout_vec)  
{  
    std::cout << "NAVIGATING FROM NODE " << path_vec->front() << " TO " << path_vec->back() << std  
        ::endl;  
    std::cout << " Path vector:";  
    print(path_vec);  
    std::cout << std::endl;  
  
    std::vector<Road> road_vec{}; //this categorizes every node in path_vec  
    for( auto it = path_vec->begin(); it < path_vec->end()-1; it++)  
    {  
        if( (*entrance_vec)[*it] )
```



UNTER

```
{  
    road_vec.push_back(Entrance);  
}  
else if( (*roundabout_vec)[*it] )  
{  
    road_vec.push_back(Roundabout);  
}  
else  
{  
    road_vec.push_back(Forward);  
}  
}  
  
int argument{};  
std::string name{};  
  
for( auto it = road_vec.begin(); it < road_vec.end(); )  
{  
    if( *it == Forward )  
    {  
        argument = 0;  
        do  
        {  
            it++;  
            argument++;  
        }  
        while( *it != Entrance && it < road_vec.end() );  
        add_forward_command(cmd_list, argument);  
    }  
    else if( *it == Entrance )  
    {  
        argument = 0;  
        do  
        {  
            it++;  
            argument++;  
        }  
        while( *it != Forward && it < road_vec.end() );  
        add_roundabout_command(cmd_list, argument-1);  
    }  
}  
add_stop_command(cmd_list);  
}  
  
void generate_full_command_list(igraph_t *graph,  
                               std::vector<Command> *cmd_list,  
                               std::vector<double> *weight_vec,  
                               int start_node,  
                               std::vector<int> destination_vector)  
{  
    // Generate the degree vector (describes the degree of each node in the network)  
    igraph_vector_t degree_vector;  
    igraph_vector_init(&degree_vector, 0);  
    igraph_degree(graph,  
                  &degree_vector,  
                  igraph_vss_all(), /*vertex_selector*/  
                  IGRAPH_OUT, /*direction*/
```



UNTER

```
-1); /* loops */

std::vector<int> path_vector{};
int end_node{};

std::vector<bool> roundabout_vector{}; //Boolean vector (the index represents each node)
std::vector<bool> roundabout_entrance_vector{}; // "-"

// Slow generation of information about the road network (cached in variables for speed later)
roundabout_vector = create_roundabout_vector(&degree_vector); // list indicating if the node is
// a roundabout or not
roundabout_entrance_vector = create_entrance_vector(graph, &degree_vector); // list indicating
// if entrance or not

while( !destination_vector.empty() )
{
    // Generate a command list for semi autonomous mode.
    path_vector = find_closest_destination(graph, weight_vec, start_node, &destination_vector);
    generate_command_list(cmd_list, &path_vector, &roundabout_entrance_vector, &roundabout_vector
);

    // Update the start & end node, remove end node from destinations
    end_node = path_vector.back();
    start_node = end_node;
    auto it = std::find(destination_vector.begin(), destination_vector.end(), end_node);
    if ( it != destination_vector.end() )
    {
        destination_vector.erase(it);
    }
}
igraph_vector_destroy(&degree_vector);
#endif
```

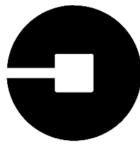
K.0.5 Engine_USART

```
/*
 * Engine_USART.c
 *
 * Created: 2019-04-01 11:36:16
 * Author: oloml269
 * Programerare: Olof Mlakar
 */

#define F_CPU 1843200UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <util/delay.h>
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU / 16) + (USART_BAUDRATE / 2)) / (USART_BAUDRATE )) - 1

///////////////////////////////
////////Global Variable Declaration////////
///////////////////////////////

volatile uint16_t default_pulsewidth = 2765;      //1.5 ms pulsewidth converted to ticks (2765)
volatile uint16_t pulsewidth_engine = 2765;         //1.5 ms pulsewidth converted to ticks (2765)
```



UNTER

```
volatile uint16_t default_pulsewidth_servo =2515; //Default steering position for the Servo in
    ticks (2515)
volatile uint16_t pulsewidth_servo = 2515;           //Default steering position for the Servo in
    ticks (2515)
volatile char mode = 0 ;                            //Default value mode
volatile int obstacle = 0;                         //Default value obstacle

//////////////////////////////Initialization of UART////////////////////////////
////////////////////////////Initialization of the diods////////////////////////////

void UART_config()
{
    // Set the baud rate registers
    UCSR0B |= (1 << RXEN0) | (1 << TXEN0); // Turn on the transmission and reception circuitry
    UCSR0C |= (1 << UCSZ00) | (1 << UCSZ01); // Set frame format: 8 data bits, 1 stop bit
    UBRROH = (BAUD_PRESCALE >> 8); // Load upper 8- bits of the baud rate value into UBRROH
    UBRROL = BAUD_PRESCALE;          // Load lower 8- bits of the baud rate value into UBBR0L
}

void UART_Transmit8(unsigned char data)
{
    while(!(UCSR0A & (1<<UDRE0)))
    {
        // Wait until buffer is empty
    }

    UDR0 = data; // Input data into transmit buffer
}

void UART_Transmit16(uint16_t data)
{
    uint8_t data_low = data & 0xff;
    uint8_t data_high = (data >> 8);

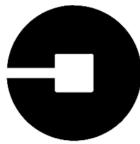
    UART_Transmit8(data_low);
    UART_Transmit8(data_high);
}

unsigned char UART_Receive8()
{
    while(!(UCSR0A & (1<<RXC0)))
    {
        // wait for data to be received
    }

    return UDR0; // Get and return received data from buffer
}

uint16_t UART_Receive16()
{
    uint8_t low_byte = UART_Receive8();
    uint8_t high_byte = UART_Receive8();
    return (high_byte << 8) | low_byte;
}

//Initialization of the diods
```



```
void lysdiod_init()
{
    DDRB |= (1<<DDB0) | (1<<DDB1);
    PORTB = (1<<PB0);
}

///////////////////////////////Initialization of Fast PWM for Engine/Servo////////////////////////////

//Initialization of the "Fast PWM"
void pwm_init_engine()
{
    DDRD |= (1<<DDD5);                                //Defines port D5 as "output"
    TCCR1A |= (1<<COM1A1) | (0<<COM1A0) | (1<<COM1B1) | (0<<COM1B0);      //Initialize "Fast PWM"
    TCCR1A |= (1<<WGM11) | (0<<WGM10);                //Waveform generation mode
    TCCR1B |= (1<<WGM13) | (1<<WGM12);
}

void pwm_init_servo()
{
    DDRB |= (1<<DDB6);                                //Defines port B6 as "output"
    TCCR3A |= (1<<COM3A1) | (0<<COM3A0) | (1<<COM3B1) | (0<<COM3B0);      //Initialize "Fast PWM"
    TCCR3A |= (1<<WGM31) | (0<<WGM30);                //Waveform generation mode
    TCCR3B |= (1<<WGM33) | (1<<WGM32);
}

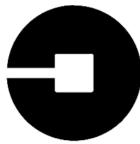
//Initiering av klockan som ska användas
void clock_init_engine()
{
    TCCR1B |= (1<<CS10);                            //Initialize the IO clocka with prescaling 1
}

//Initiering av klockan som ska användas
void clock_init_servo()
{
    TCCR3B |= (1<<CS30);                            //Initialize the IO clocka with prescaling 1
}

///////////////////////////////Initialization of External Interrupts////////////////////////////

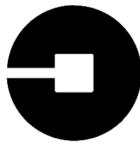
void set_external_interrupt()
{
    sei(); /* Set Global Interrupt Enable (sei comes from avr/interrupt.h) */ // _enable_interrupt();
    EICRA |= (0<<ISC01) | (1<<ISC00); // Trigger on any edge
    EIMSK |= (1<<INT0);           //Enable interrupts
}

//External interrupt
ISR(INT0_vect)
{
    if(mode == 1) //Cheks witch mode is active (Manual, Semi- or Autonomus)
    {
        if(obstacle == 1) //If in mode "manual" and detects an obstacle, BREAK
    }
}
```



UNTER

```
{  
    pulsedwidth_engine = 2765;  
    OCR1A = pulsedwidth_engine;  
    pulsedwidth_servo = default_pulsedwidth_servo;  
    OCR3A = pulsedwidth_servo;  
    obstacle = 0;  
}  
else  
{  
    obstacle = 1;  
}  
}  
else  
{  
    //If in mode "Semi- or Autonomus" and detects an obstacle, BREAK  
}  
{  
    OCR1A = 2765;  
    OCR3A = 2515;  
}  
}  
  
void set_D2_input()  
{  
    DDRD |= (0<<DDD2); //Configures PD2 as input an pin  
}  
  
//////////////////////////////  
//////////Steering functions for Engine/Servo//////////  
/////////////////////////////  
  
////Checks if the pulsedwidth values for the engine and servo are between 1.0 ms and 2.0 ms///  
uint16_t checkduty(uint16_t duty, uint16_t maxduty, uint16_t minduty)  
{  
    if(duty>maxduty || duty<minduty)  
    {  
        printf("Requested duty cycle too large. Maximum possible duty cycle is 2ms\n");  
        return 0;  
    }  
    return 1;  
}  
  
uint16_t checkduty_max(uint16_t duty, uint16_t maxduty)  
{  
    if(duty>maxduty)  
    {  
        printf("Requested duty cycle too large. Maximum possible duty cycle is 2ms\n");  
        return 0;  
    }  
    return 1;  
}  
  
uint16_t checkduty_min(uint16_t duty, uint16_t minduty)  
{  
    if(duty<minduty)  
    {  
        printf("Requested duty cycle too small. minimum possible duty cycle is 1ms\n");  
    }  
}
```



UNTER

```
    return 0;
}
return 1;
}

void init_motor(uint16_t default_val)
{
    OCR1A = default_val; // Set the pulsedwidth to 1.5ms
}
void init_servo(uint16_t default_val)
{
    OCR3A = default_val; // Set the pulsedwidth to 1.5ms
}

void forward(uint16_t default_val)
{
    OCR1A = default_val + 80; // Command for driving forward
}

void backwards(uint16_t default_val)
{
    OCR1A = default_val - 150; // Command for driving backwards
}

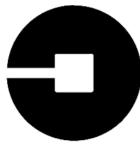
void stop(uint16_t default_val)
{
    OCR1A = default_val - 200;
    _delay_ms(300);
    OCR1A = default_val; // Command for breaking
}

void drive_straight(uint16_t default_val)
{
    OCR3A = default_val ; // Command for driving straight
}

void turn_left(uint16_t default_val)
{
    OCR3A = default_val - 300; // Command for turning left
}

void turn_right(uint16_t default_val)
{
    OCR3A = default_val + 300; // Command for turning right
}

///////////////////////////////Main Function///////////////////////////////
///////////////////////////////Main Function/////////////////////////////
///////////////////////////////Main Function/////////////////////////////
int main(void)
{
    char input;
    uint16_t command_code;
    int servo_UART;
    uint16_t clocktop = 36863; //Defines the Period and sets it to 20 ms in
                                //Ticks (36863)
```



UNTER

```
uint16_t maxduty = 0.095*clocktop;           //Period 20ms, max 2ms
uint16_t minduty = 0.05*clocktop;            //Period 20ms, min 1ms
uint16_t engine_max = 0.08*clocktop;          //Practicel max engine speed
ICR1 = clocktop;
ICR3 = clocktop;
//////////Running the initialization functions/////////
UART_config();
init_motor(default_pulsewidth);
init_servo(default_pulsewidth);
pwm_init_engine();
pwm_init_servo();
clock_init_engine();
clock_init_servo();
lysdiod_init();
set_external_interrupt();
set_D2_input();

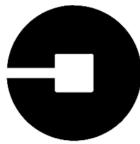
mode = UART_Receive8();           //Checks witch mode is selected from the RPI
PORTB &= ~(1<<DDB0);

switch(mode) {
    case 1 :
        /////////////////////////////////
        //////////////////Manuel mode////////////////
        /////////////////////////////////
    while(1)
    {
        // Motor loop
        PORTB &= ~(1<<PB1);
        if(checkduty_max(pulsewidth_engine, engine_max) == 1)
        {
            OCR1A = pulsewidth_engine;
        }
        else
        {
            OCR1A = engine_max;
        }

        if(checkduty(pulsewidth_servo, maxduty, minduty))
        {
            OCR3A = pulsewidth_servo;
        }
        else if(pulsewidth_servo > maxduty)
        {
            OCR3A = maxduty - 20;
            pulsewidth_servo=OCR3A;
        }
        else
        {
            OCR3A = minduty + 20;
            pulsewidth_servo=OCR3A;
        }

        // UART loop
        input = UART_Receive8();

        switch(input){
```



UNTER

```
    case 1 :
        pulsewidth_engine = pulsewidth_engine + 10;      // Starts the engine and increases the
speed gradually (1)
        break;
    case 2 :
        stop(default_pulsewidth);
        pulsewidth_engine = default_pulsewidth; // Turn off engine (2)
        break;
    case 3 :
        pulsewidth_servo = pulsewidth_servo - 90; // Turns left gradually (3)
        break;
    case 4 :
        pulsewidth_servo = pulsewidth_servo + 90; // Turns right gradually (4)
        break;
    case 5 :
        pulsewidth_servo = default_pulsewidth_servo; // Default servo angle (5)
        break;
    }

    UART_Transmit16(OCR1A);
    UART_Transmit16(OCR3A);

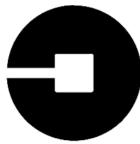
}

case 2 :
//////////////////////////////Semi-autonomous mode////////////////////////////
OCR3A = default_pulsewidth_servo;
while(1)
{
    command_code = UART_Receive16();
    switch(command_code){
        case 11 :
            forward(default_pulsewidth); // Drive forward (11)
            break;
        case 12 :
            forward(default_pulsewidth); // Drive in roundabout (12)
            break;
        case 13 :
            stop(default_pulsewidth); // Stop (13)
            break;
    }

    while(1)
    {
        servo_UART = UART_Receive16();

        if(servo_UART == 75) // Checks if the RPI sends an ok-signal (75)
        {
            break; // Breaks out from the LOOP
        }

        if(checkduty(servo_UART, maxduty, minduty))
        {
            OCR3A = servo_UART;
        }
    }
}
```



UNTER

```
else if(servo_UART > maxduty)
{
    OCR3A = maxduty - 20;
}
else
{
    OCR3A = minduty + 20;
}
}

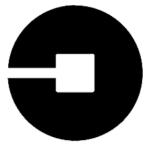
case 3 :
//////////Autonomous mode///////////
OCR3A = default_pulsewidth_servo;
while(1)
{
    command_code = UART_Receive16();
    switch(command_code){
        case 11 :
            forward(default_pulsewidth);      // Drive forward (11)
            break;
        case 12 :
            forward(default_pulsewidth);      // Drive in roundabout (12)
            break;
        case 13 :
            stop(default_pulsewidth);         // Stop (13)
            break;
    }

    while(1)
    {
        servo_UART = UART_Receive16();

        if(servo_UART == 75) // Checks if the RPI sends an ok-signal (75)
        {
            break; // Breaks out from the LOOP
        }

        if(checkduty(servo_UART, maxduty, minduty))
        {
            OCR3A = servo_UART;
        }
        else if(servo_UART > maxduty)
        {
            OCR3A = maxduty - 20;
        }
        else
        {
            OCR3A = minduty + 20;
        }

    }
}
}
```

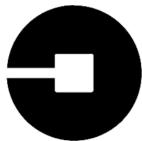


Autonom taxibil

2021–07–09

UNTER

J ANVÄNDARMANUAL



UNTER

Autonom taxibil

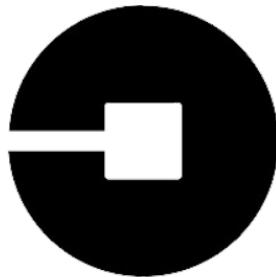
4 juni 2019

Användarmanual

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl

4 juni 2019

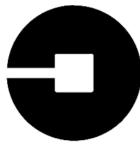
Version 1.0



UNTER

Status

Granskad	Nicholas Sepp	2019-05-22
Godkänd		2019-xx-xx

**UNTER****Projektidentitet**

Grupp E-post: jonca673@student.liu.se

Hemsida: <http://wwwisy.liu.se/edu/kurs/TSEA56/>

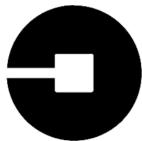
Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

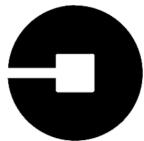
Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

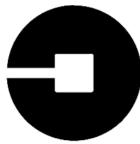
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Datorseendeansvarig (DAT)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se

**UNTER****INNEHÅLL**

1	Inledning	1
2	Produkt	1
3	Krav på omgivning	1
4	Användning	2
4.1	Uppstart av bilen	3
4.2	Kalibrering av datorseende	3
4.3	Start av huvudprogram	3
4.3.1	Manuellt läge	4
4.3.2	Semi-autonomt läge	4
4.3.3	Autonomt läge	5
5	Styrkor	5
6	Begränsningar	6

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda ändringar	Utförda av	Granskad
1.0	2019-05-22	Version 1.0	Grupp 9	NS



UNTER

1 INLEDNING

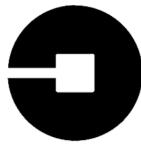
Bilen är en autonom taxibil som kan utföra köruppdrag utifrån en tillhandahållen karta. Köruppdragen består av att bilen från given startpunkt tar sig till angivna destinationer och stannar. Med hjälp av datorseende kan bilen navigera i vägnätet och detektera samt stanna vid stopplinjer. Bilen kan utföra sina uppdrag genom att köras manuellt, semi-autonomt samt autonomt. Manuell körning sker av att användaren styr bilen från tangentbordet. Semi-autonom körning sker av att användaren skriver in körkommandon till programmet. Autonom körning sker av att användaren skriver startpunkt och destinationer till programmet.

2 PRODUKT

Produkten består av en radiostyrd bil där en Raspberry PI 3, en Raspberry PI kamera modul och ett virkort med två ATmega1284p mikrodatorer är monterade. Två halleffektsswitchar samt en ultraljudssenor finns även monterade på bilen. Användaren kopplar upp sig med hjälp av en bärbar dator till Raspberry PI enheten och därifrån ges kördirektiv för bilen. Bilen kan köras manuellt med hjälp av W, A, S, D tangenterna på den bärbara persondatorn. Bilen kan även köras semi-autonomt och autonomt, vid dessa två lägen följer bilen vägen med hjälp av datorseende. Vid semi-autonom körning skriver användaren in körkommandon som bilen sedan utför. Vid autonoma körning skickas ett vägnät och önskade destinationer till bilen som löser ett kortaste väg problem för att sedan på kortast möjliga väg köra till de angivna destinationerna.

3 KRAV PÅ OMGIVNING

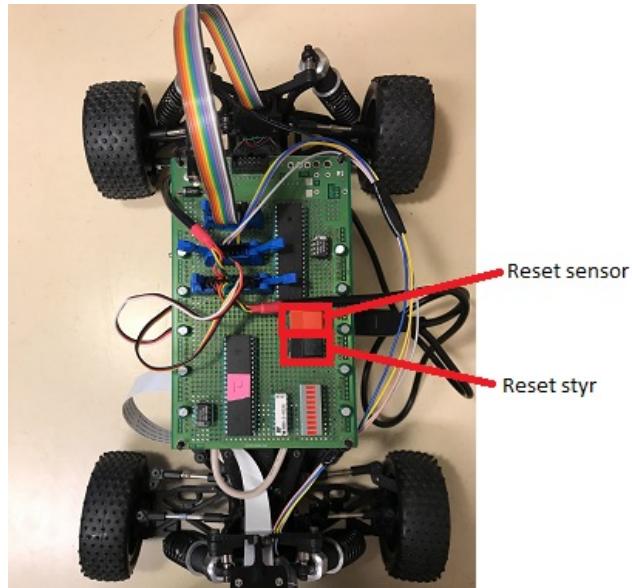
Bilen är designad för att genomföra ett köruppdrag i salen Visionen med en förinställd bakgrundsbelysning på ljus-nivå 3 från taklamporna. Köruppdraget ska genomföras på en projicerad väg från två projektorer monterade i taket. Vägnätet ska vara tvåfiligt och kan innehålla raksträckor, svängar, rondeller och stopplinjer. Bilens destinationer får ej placeras inuti rondeller. För vidare banspecifikationer se dokument, Ban och tävlingsregler.



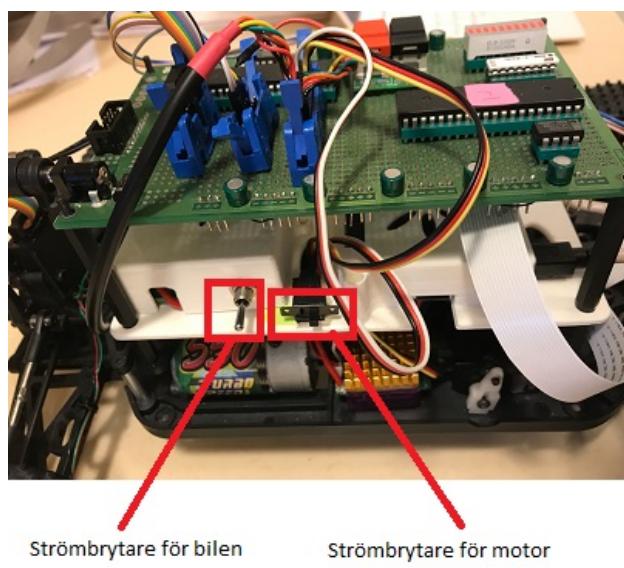
UNTER

4 ANVÄNDNING

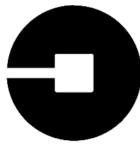
Detta stycke beskriver hur bilen är tänkt att användas för att inga problem ska uppstå. I figur 1 nedan syns bilens två resetknappar för styr- och sensormodulen. I figur 2 ses de två strömbrytarna för bilen.



Figur 1: Bilens två reset knappar



Figur 2: Bilens två strömbrytare



UNTER

4.1 Uppstart av bilen

Uppstarten av bilen utförs i följande steg.

1. Se till att batteriet är fullladdat innan start.
2. Verifiera att båda UART-kablarna är korrekt monterade. UART-kabeln till styrAVR ska vara monterad i den övre USB porten och UART kabeln till sensorAVR:en ska vara monterad i den undre USB-porten.
3. För strömbrytaren för bilen uppåt för att slå på strömmen för bilen.
4. För strömbrytaren för motorn mot den grönmärkade sidan för att starta bilens motor.
5. Med persondatorn anslut dig till Raspberry PI 3 med hjälp av ssh.
6. På persondatorn starta sedan VNC för att få tillgång till operativsystemets användargränssnitt på Raspberry PI:n.

4.2 Kalibrering av datorseende

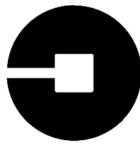
Färgdetektionen är känslig för miljön i visionen och kan påverkas av dagsljus som lyser in i visionen. Om ljusnivån i visionen ändras tillräckligt mycket kan datorseendet behövas omkalibreras. Orsakerna kan vara tid på dygn, väder samt att närliggande rum är tända eller släckta.

- 1.Verifiera att stopplinje detektionen har slutat fungera genom att ställa bilen vid en blå linje och kontrollera att den inte detekteras.
- 2.Öppna koden för kalibreringsprogrammet för färgmodellskalibrering och ställ in parametrar så endast blåa linjer detekteras.
- 3.Välj den bästa modellen och spara vald modells färgrymd-parametrar.
- 4.Öppna huvudprogrammet och programmera in valda färgrymdsparametrar för stopplinjedetektion.
- 5.Spara och kompilera koden.

4.3 Start av huvudprogram

När väl datorseendet är kalibrerat kan huvudprogrammet köras.

1. Starta huvudprogrammet.
- 2.Vid uppstart kan tre körlägen väljas: Manuellt, Semi-autonotm eller Autonomt. Val av läge sker genom att på tangentbordet trycka på M för manuellt, S för semi-autonotm eller A för autonomt.
- 3.Verifiera att ATmega1284p till styrmodulen är rätt programmerad. Detta görs genom att i manuellt läge ge fullt styrutslag åt höger. Skulle hjulen vid fullt styrutslag hoppa tillbaka till ursprungsläget så behöver styrmodulens ATmega1284p programmeras om. Gå då till steg 4, annars hoppa över och gå till steg 5.
- 4.Programmera om ATmega1284p för styrmodul genom att starta Atmel Studio och programmera in koden för styrmodulen med hjälp av JTAG.



UNTER

5. Stäng av programmet och nollställ både ATmega1284p för senosormodulen samt styrmodulen. Detta görs genom att trycka på de två knapparna på virkortet.
6. Verifiera att lysdioden som är närmast bilens för lyser. Detta betyder att bilen är i mode select läge och redo att ta emot val av körläge.
7. Starta programmet och välj körläge genom tangentbordskommando. M för manuellt läge, S för semi-autonomt läge eller A för autonomt läge.

4.3.1 *Manuellt läge*

För att aktivera manuellt läge för bilen anger användaren kommandot M i bilens startsekvens, se sektion 4.3. I manuellt läge har användaren själv full kontroll över bilen genom att använda tangentbordet. De tangenter som används för att kontrollera bilen i manuellt läge finns i tabell 1.

Tabell 1: Tangenter för kontroll av bilen vid manuellt läge

Tangent	Funktion
W	Öka gaspådrag
S	Minska gaspådrag
A	Ändra styrutslag i vänster riktning
D	Ändra styrutslag i höger riktning
Mellanslag	Stanna bilen

Vid start av manuellt läge finns risken att bilen inte ger något gaspådrag. Om detta sker behöver bilens styr och sensor AVR resettas genom att trycka på de två knapparna på virkortet.

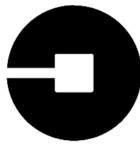
4.3.2 *Semi-autonomt läge*

I semi-autonomt läge har användaren inte full kontroll utav bilen utan förser bilen med körkommandon. Användaren anger körkommandon efter det att semi-autonomt läge valts genom att skriva in de kommandon som finns i tabell 2. Det finns ingen satt gräns för hur många körkommandon som kan skickas till bilen utan det begränsas endast utav bilens tillgängliga programminne.

Tabell 2: Kördirektiv för semi-autonomt läge

Direktiv	Kommando	Argument
Vanlig väg	forward	X
Rondell	roundabout	Y
Destination	stop	1
Slutfört uppdrag	done	1

Kördirektivet vanlig väg representeras av körkommandot forward och tillåter bilen att nyttja dess regleralgoritmer för körning på vanlig väg. Direktivet avslutas med ett argument X, där X är ett heltal som representerar hur många stopplinjer bilen ska passera innan nästa körkommando ska utföras. För körning i rondell anges körkommandot roundabout följt av argumentet Y, där Y är ett heltal som representerar vilken rondellavfart bilen ska ta. För att stanna vid en destination anges kommandot stop 1. Kommandot done 1 markerar att uppdraget är slutfört och stoppar programmet.



UNTER

Nedan finns ett exempel på semi-autonom körning.

Körkommandon som skickas till bilen:

```
forward 2  
roundabout 2  
stop 1  
forward 3  
stop 1  
done 1
```

Bilen kommer därefter att utföra ett köruppdrag. Exempeluppdraget ovan består av att bilen först kommer köra förbi en stopplinje och vid andra linjen komma in i en rondell. I rondellen kommer bilen att ta den andra avfarten för att sedan stanna vid första nästkommande stopplinje. Bilen kommer därefter fortsätta till nästa destination där den kommer att åka förbi två linjer och stanna vid den tredje. Vid den andra destinationen har bilen utfört sitt uppdrag och programmet kommer att avslutas. Skulle inte bilen ge något gaspådrag har det inträffat en bugg. Denna bugg kallas för det klassiska felet och användaren behöver då stänga av programmet samt starta om de båda ATmega1284p genom att trycka på dess båda nollställningsbrytare för att sedan starta om programmet, välja semi-autonoms körläge och mata in kommandolistan. Detta upprepas till att bilen genererar gaspådrag. När bilen har utfört ett köruppdrag stannar den vid sista destinationen och programmet stängs därefter av.

4.3.3 Autonoms läge

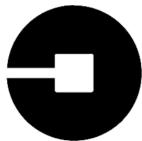
I autonoms läge utför bilen köruppdrag helt på egen hand. Användaren förser bilen med en karta över vägnätet som är ett enkelriktat nodnät. Efter att kartan skickats till bilen anger användaren en startposition för bilen som är den nod i vägnätet bilen är placerad vid start av köruppdraget samt ett antal destinationer som ska besökas under köruppdraget. Startpositionen kan inte vara en destination som ska besökas och en destination kan inte besökas två gånger. Nedan beskrivs hur ett autonoms körläge påbörjas.

1. Skriv in bilens startnod
2. Skriv in bilens destinationsnoder
3. Bilen börjar sitt köruppdrag tills den har besökt alla destinationer varpå programmet avslutas.

Ibland startas inte motorn när ett köruppdrag påbörjas. Användaren måste då resetta styr- och sensormodulen genom att trycka på de två knapparna på virkortet.

5 STYRKOR

Programmet är skapat till att vara väldigt generellt och fungera i Visionen oavsett påverkan av omgivningen. Genom att ha möjlighet till att kalibrera om datorseendet klarar bilen av att detektera väglinjerna väldigt väl. Datorseendet har haft genomgående tester och dess implementation är robust. Styralgoritmerna för körkommandona är även de skrivna på ett generellt manér att regleras efter datorseendet. Det finns ingen så kallad ”hårdkodad” implementation som löser problematiska situationer, exempelvis kraftig sväng i en rondell. Kortaste vägen algoritmen baseras på att vägkartan

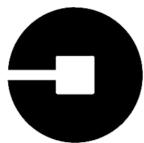


UNTER

skapar ett nodnätverk och skapar sedan körkommandon för kortaste vägen till destinationer. Har man en gång genererat ett nodnätverk kommer programmet alltid att ta fram den kortaste vägen till vald destination. Programmet är således en god grund för en elektronikintresserad individ att vidareutveckla.

6 BEGRÄNSNINGAR

Då programmet inte använder sig av något användarvänligt gränssnitt, att man beroende på situationen i Visionen kan behöva ändra på källkoden för datorseende och att man i och med buggar behöver programmera om ATmega1284p krävs en viss form av elektronikkunskap.



Autonom taxibil

2021–07–09

UNTER

K EFTERSTUDIE



UNTER

Autonom taxibil

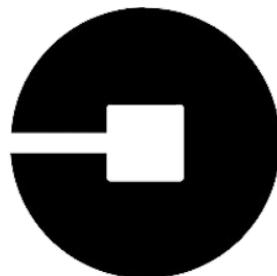
5 juni 2019

Efterstudie

Jonathan Carlin, Olof Mlakar, Emil Mårtensson, Nicholas Sepp och Dennis Österdahl

5 juni 2019

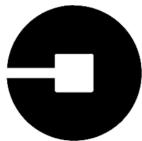
Version 1.0



UNTER

Status

Granskad	Nicholas Sepp	5 juni 2019
Godkänd		5 juni 2019

**UNTER****Projektidentitet**

Grupp E-post: jonca673@student.liu.se

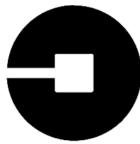
Beställare: Anders Nilsson, Linköpings universitet
Tfn: +46 13282635
E-post: anders.p.nilsson@liu.se

Handledare: Olov Andersson
Tfn: +46 13282658
E-post: olov.andersson@liu.se

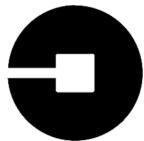
Kursansvarig: Mattias Krysander
Tfn: +46 13282198
E-post: mattias.krysander@liu.se

Projektdeltagare

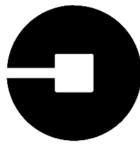
Namn	Ansvar	E-post
Jonathan Carlin	Projektledare (PL)	jonca673@student.liu.se
Olof Mlakar	Dokumentansvarig (DOK)	oloml269@student.liu.se
Emil Mårtensson	Testansvarig (TEST)	emima951@student.liu.se
Dennis Österdahl	Datorseendeansvarig (DAT)	denos835@student.liu.se
Nicholas Sepp	Kvalitetssamordnare (QA)	nicse725@student.liu.se

**UNTER****INNEHÅLL**

1	Tidsåtgång	1
1.1	Arbetsfördelning	1
1.2	Tidsåtgång jämfört med planerad tid	2
2	Analys av arbete och problem	2
2.1	Vad hände under de olika faserna	2
2.2	Hur vi arbetade tillsammans	2
2.3	Hur använde vi projektmodellen	3
2.4	Hur fungerade relationen med beställaren	3
2.5	Hur fungerade relationen med handledaren	3
2.6	Tekniska framgångar och motgångar	3
2.6.1	Framgångar	3
2.6.2	Motgångar	3
3	Måluppfyllelse	4
3.1	Vad har uppnåtts	4
3.2	Hur fungerade leveransen	4
3.3	Hur har studiesituationen påverkat projektet	4
4	Sammanfattning	4
4.1	De tre viktigaste erfarenheterna	4
4.2	Goda råd till de som ska utföra liknande projekt	5

**UNTER****DOKUMENTHISTORIK**

Version	Datum	Utförda ändringar	Utförda av	Granskad
1.0	5 juni 2019	Första utkast	Grupp 9	NS



UNTER

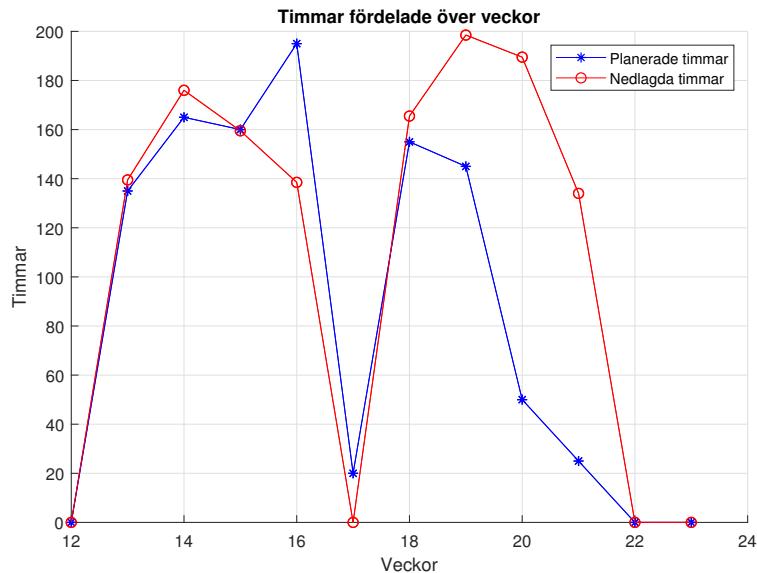
1 TIDSÅTGÅNG

Följande kaptiel beskriver hur projektgruppen har lagt upp sina arbetstimmars och hur de har fördelats över alla projektmedlemmar.

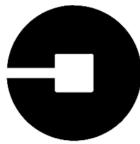
1.1 Arbetsfördelning

Gruppen inledder i början av projektet en planering av arbetstimmarna för underfasen. Figur 1 nedan visar hur timmarna fördelades över projektveckorna. Gruppen valde att fram till och med vecka 17 öka sina arbetstimmars för att under vecka 17 ta ledigt för påsk och omtentor. Efter vecka 17 skulle gruppen sakta öka upp sina timmar för att vid vecka 19 minska dessa. Tidsplanering höll sig väldigt bra fram till och med vecka 17. Vad som kom att hänta var att mot slutet av projektet behövde mer tid läggas än planerat. Detta då gruppen hade förberett sitt datorseende för salen Muxen och inte salen Visionen. Hade gruppen tidigare utvecklat sitt datorseende i Visionen hade troligen de lagda timmarna blivit något färre.

Gruppen lade upp sin arbetstid att vara på plats och kontinuerligt arbeta vardagar mellan 8-17. Arbetsuppgifterna delades upp lika så att alla skulle arbeta lika många timmar. Självklart blir det alltid differens mellan lagda arbetstimmars. Gruppen har diskuterat det och kommit fram till att är någonting som är naturligt och att man inte kan vara exakt lika på sekunden.



Figur 1: Fördelning av arbetstimmars. Röd graf representerar antal nedlagda timmar, blå graf representerar antal planerade timmar.



UNTER

1.2 Tidsåtgång jämfört med planerad tid

Tabell 1 nedan redogör för den planerade tiden och den använda tiden för de olika faserna under projektets gång. Förefasen innehöll planering, förstudie och designspecifikation. Underfasen var själva fasen då konstruktionen av taxibilen utfördes. Efterfasen bestod av planering av presentation, opponering, skrivande av användarhandledning och efterstudie.

Tabell 1: Tidsåtgång jämfört med planerad tid

Fas	Planerad tid i timmar	Använt tid i timmar
Före	Krav ca 60 tim, Plan ca 100 tim	200,5
Under	Ca 1150 timmar	1433,5
Efter	Ca 60 timmar	61,5

2 ANALYS AV ARBETE OCH PROBLEM

Under arbetet med projektet uppkom ett antal problem och situationer som behövde överkommas. I följande rubriker beskrivs hur projektgruppen arbetade för att förhindra problem, vilka problem som uppstod och hur dessa problem lösades.

2.1 Vad hände under de olika faserna

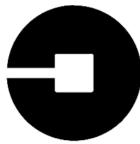
I projektets förefas arbetade projektgruppen med att skapa en bra grundutförande av projektet. En projektplan och en tidplan författades för att få en överblick om hur projektet planerades att genomföras och en tidplan skapades för att fördela tiden av projektets underfas på ett bra sätt. Projektgruppen spenderade mycket tid med att skriva en designspecifikation för att underlätta genomförandet av projektets praktiska arbete. Detta för att minimera problem som kunde ha uppstått på grund av felaktigt produktdesign och missförstånd angående produktens konstruktion.

I projektets underfas konstruerades den bil som designats i projektets före fas. Vid konstruktionen uppstod vissa problem då den tänkta implementationen inte fungerade som planerat. Projektgruppen överkom dessa problem genom att felsöka problemet och designa om implementationen på så sätt att en ny fungerande implementation erhölls.

I projektets efterfas skrevs en användarmanual för hur bilden används på korrekt sätt. En presentation av den färdiga produkten hölls för beställare och motståndare och projektgruppen gjorde en motivering på en annan projektgruppars arbete.

2.2 Hur vi arbetade tillsammans

Gruppen arbetade parallellt då det gick, ofta två och två med någon som kunde gå mellan paren eller i grupper av två och tre. Projektledaren hade huvudansvaret att dela ut arbetsuppgifter och strukturera upp gruppens arbete. Detta gjordes främst med avstamp i milstolparna och aktiviteterna som formulerats i projektplanen. Gruppmedlemmarna uppdaterade varandra kontinuerligt om i vilket läge deras nuvarande arbetsuppgift var i så att hela gruppen hade insikt i var gruppen var relativt milstolparna.



UNTER

2.3 Hur använde vi projektmodellen

Projektmodellen användes genom att samtliga dokument är skrivna efter LIPS-mallen och projektet har genomförts enligt LIPS-modellen.

2.4 Hur fungerade relationen med beställaren

Relationen med beställaren har fungerat bra. Mycket av den formella kommunikationen har förts via mejl men gruppen har även haft möjlighet att diskutera saker på tu man hand med beställaren samt vid så kallade beställarmöten.

2.5 Hur fungerade relationen med handledaren

Relationen med handledaren fungerade väl. Gruppen hade veckovis möten med handledaren och stämde av både framsteg och de problem som behövde lösas. Gruppen kände att de ofta kunde få kontakt och hjälp av handledaren. Detta gav en betryggande känsla.

2.6 Tekniska framgångar och motgångar

Under projektets gång har projektgruppen haft flera framgångar och motgångar. Nedan presenteras de tre största framgångarna och motgångarna som projektgruppen haft.

2.6.1 Framgångar

Gruppen hade stora framgångar de första veckorna av underfasen. Mycket förarbete som var vitalt för projektets utveckling, som implementering av den trådlösa kommunikationen mellan bilen och persondatorn samt fungerande manuell styrning, skedde utan några större motgångar.

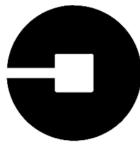
Gruppen hade även stor framgång med att parallellisera arbetet vilket gjorde att projektets underfas flöt på bra. Då bilens olika sensorer och funktioner kunde utvecklas och testas parallellt för att sedan sättas samman har gruppen tagit fram en nödbroms som aktiveras vid upptäckt av hinder på vägen. Gruppens produkt har även en välfungerande Halleffektssensor som mäter upp bilens körningssträcka med en säkerhet på 2,6 cm.

Slutligen så kunde bilens kortaste-väg-algorithm implementeras utan några motgångar och tack vare den funktionen kunde bilen utföra sina köruppdrag på kortast möjliga sträcka.

2.6.2 Motgångar

Projektgruppen har haft kontinuerliga problem med det datorseende som implementerats för att detektera väg och stopplinjer. Miljön i salen Visionen varierar mycket vilket medförde att parametrar för väg- och stopp-linje detektion ofta behövde kalibreras om. Detta medförde att en implementation som fungerade inte garanterat fungerade nästa dag om miljön förändrats.

En annan motgång projektgruppen stötte på var bilens reglering. Bilens reglering baserar sig på de väglinjer som datorseendet detekterar vilket inte alltid görs. Detta medför att regleringen måste reglera på en signal som inte alltid existerar och detta försvårade problemet.



Busskommunikationen på bilen var en tredje motgång projektgruppen hade vid projektets genomförande. Det tog tid innan busskommunikationens funktionalitet var fastställd vilket försvårade integrationen av bilens olika moduler något.

3 MÅLUPPFYLLELSE

I följande rubriker beskrivs vilka mål projektgruppen har uppnått vid utförandet av projektet.

3.1 Vad har uppnåtts

Projektgruppen har uppnått majoriteten av uppställda krav med prioritet 1.

3.2 Hur fungerade leveransen

Under leveransen fanns fortfarande bekymmer med datorseende i Visionen. Efter omkalibrering och testkörningar lyckades bilen fullfölja ett par köruppdrag. Gruppen känner sig även nöjda med sin framläggning och opponering.

3.3 Hur har studiesituationen påverkat projektet

Mycket tid har gått åt till projektet och många i gruppen känner att övriga kurser har kommit i andra hand.

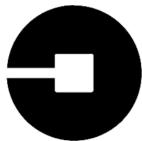
4 SAMMANFATTNING

I det stora hela är projektgruppen nöjd med genomförandet av projektet och det erhållna resultatet. Projektet har varit mycket lärorikt och givit en bra inblick i vad att projektarbete innebär och hur ett större projekt genomförs. I nedanstående rubriker presenteras de erfarenheter projektgruppen tar med sig från kursen och de råd projektgruppen har till de som genomför liknande projekt i framtiden.

4.1 De tre viktigaste erfarenheterna

Gruppen kommer framförallt ta med sig tre erfarenheter från projektet:

- Arbeta tillsammans.
- Parallelisera arbetet så mycket som möjligt.
- Börja testfasen tidigt och i rätt miljö.



UNTER

4.2 Goda råd till de som ska utföra liknande projekt

Börja i tid, det är lätt att skjuta upp saker. Ett enkelt sätt att motverka detta är att sätta upp milstolpar i projektet. Se till att ha en milstolpe varje vecka för att ha ett mål att jobba mot men se till att inte ha för många då det blir stressigt om någon av milstolparna inte uppfylls.

Se till att ha god kommunikation inom gruppen. Konflikter och oenigheter kommer att uppstå och utan god kommunikation mellan gruppmedlemmarna kommer det vara svårt att ta sig fram.

Parallelisera arbetet inom gruppen. Se till att alla har något att arbeta med för att undvika flaskhalsar och för att få en bättre fördelning på arbetet.

De projicerade vägarna i Visionen presenterar helt andra förutsättningar för datorseendet än traditionella vägar. Arbeta med datorseendet på vanliga vägar, t.ex. vägar i form av svart tejp på vit papp, kommer nödvändigtvis inte kunna överföras till ett fungerande datorseendet i Visionen. Börja utveckla datorseendet för Visionens projicerade vägar tidigt.

Glöm inte att ha roligt. Projektet må vara väldigt tidskrävande och kännas svårt men det är väldigt lärorikt, spännande och roligt att få testa sina kunskaper och konstruera något.