CHALMERS
UNIVERSITY OF TECHNOLOGY

Implementering av styrfunktioner i touchdisplay för bussar

Implementation of control functions in touch display for buses

Examensarbete inom högskoleingenjörsprogrammet Elektroteknik

Andreas Pettersson

Adam Östensson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2017

Examensarbete 2017

Implementering av styrfunktioner i touchdisplay för bussar

Implementation of control functions in touch display for buses

Andreas Pettersson
Adam Östensson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2017

Implementering av styrfunktioner i touchdisplay för bussar
Implementation of control functions in touch display for buses
Andreas Pettersson
Adam Östensson

Handledare: Peter Lundin, Data- och Informationsteknik
Examinator: Christer Carlsson, Data- och Informationsteknik

Examensarbete 2017
Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
SE-412 96 Gothenburg
Telefonnummer +46 31 772 1000

Omslag: Slutprodukt med klimatpanel som aktiv sida

Typeset in LaTeX
Göteborg, Sverige 2017

Förord

Examenarbetet är utfört av två högskoleingengörer med inriktning Elektroteknik vid Chalmers Tekniska Högskola och omfattar 15 HP. Projektet är utfört under Benteler Engineering Services med inriktning mot bussar. Arbetet har till största del utförts på företagets kontor på Nya varvet, Västra Frölunda. Tålamod, ihållighet och mod är ord som sammanfattar de lärorika veckorna vi har fått ta del av. Vi vill förutom våra medarbetare på Benteler Engineering rikta ett särskilt stort tack till våra handledare:

- David Anberg, Benteler Engineering Services AB
- Peter Lundin, Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola.

Andreas Pettersson & Adam Östensson, Göteborg, December 2016

Sammanfattning

Mekaniska knappar står för majoriteten av det man kan styra i dagens bussar, allt från att höja ljudet på radion till att ändra temperaturen för passagerare. Dessa knappars placering kan variera beroende på vilken buss man blir tilldelad, och innebörden av knappen kan verka otydlig. Projektets uppgift var att ta fram en prototyp som ersätter mekaniska knappar med en touchdisplay med tillhörande styrenhet. Touchdisplayen strävar efter att vara lättare att manövrera, både ur ett operativt som ergonomiskt perspektiv, jämfört med mekaniska knappar. Införandet av touchdisplay kan även leda till minskade kostnader i och med att tillverkningen och installation av knappar försvinner. Arbetet har omfattat val av display, utveckling av hårdvara och mjukvara för styrenheten samt resulterat i en funktionell prototyp med ett såväl modernt som användarvänligt gränssnitt. Prototypen har avgränsat sig till funktionerna temperaturreglering och ljusbelysning med utrymme för vidareutveckling. Prototypen har utformats utifrån olika standarder, tidigare examensarbete, krav och kompletterade önskemål från Volvo Bussar, samt författarnas egna idéer. Projektet har utförts på Benteler Engineering Services AB i Göteborg.

Nyckelord: Raspberry Pi, Ergonomi, Gränssnitt, Buss, Python, HMI

Abstract

Mechanical buttons stands for the majority of control functions that can be controlled in today´s buses, everything from increasing the volume to regulate the temperature for passengers. The position of the mechanical buttons can vary depending on the bus, and the meaning of each button may seem unclear. The project task was to develop a prototype that replaces mechanical buttons with a touch display connected to a control unit. The touch display strives to be easier to maneuver, both from an operational and ergonomic perspective, compared to mechanical buttons. The use of a touch display can also lead to reduced costs as the manufacture and installation of buttons disappear. The work has included choice of display, development of hardware and software for the control unit and resulted in a functional prototype of a modern and user-friendly interface. The prototype has been limited to the functions temperature control and interior lightning with room for further development. The prototype has been designed from standards, other thesis work, demands and wishes from Volvo Buses, and also by the authors' ideas. The project has been done at Benteler Engineering Services AB in Gothenburg.

Keywords: Raspberry Pi, Ergonomic, Interface, Bus, Python, HMI

Innehåll

Beteckningar

3D - Tredimensionell
Benteler - Benteler Engineering Services AB
CAN - Controller Area Network
C - Programmeringsspråket C
DB9 - Seriell kontakt med 9 pinnar
GPIO - General-purpose input/output
Gradient - Färgskiftning beroende av temperatur (varmt till kallt - röd till blått)
HMI - Human Machine Interface
Haptisk feedback - Återkoppling i form av vibration
HDMI - High-Definition Multimedia Interface
I/O - Input/Output
OBD2 - On-board-Diagnistics, diagnosuttag
PICAN - Extern CAN-kompatibel modul
Python - Programmeringsspråket Python
SPI - Serial Peripheral Interface
USB - Universal Serial Bus

Figurer

Figurer utan källhänvisning är skapta av författarna själva

Tabeller

1
Inledning

1.1 Bakgrund

Benteler har sålt in en idé till Volvo Bussar om att implementera en touchdisplay som ska ersätta befintliga hårdvarufunktioner som styr till exempel temperatur och belysning. Genom att ersätta fysiska knappar med en touchdisplay kan kostnader minskas i produktionen av instrumentpanel. Det ska också underlätta för chaufförен att ändra inställningar.

Prototypen ska vidareutvecklas med avseende på tidigare examensarbete *Prototyp för duplicering av instrumentpanel anpassad för övningskörning av buss* [27]. Detta examensarbete utvecklade och konstruerade en prototyp för övningsförare och hade syftet att underlätta övervakning vid övningskörning. Från examensarbete *Omdesign av klimatpanel i Volvos bussar* [26], som utförde ett analysarbete om placering av klimatpanel för bussar, gavs stödjande information om klimatpanelens utformning.

1.2 Syfte

Uppdragets syfte är att ta fram en prototyp bestående av en touchdisplay och styrenhet som kan monteras i en buss och reglera flertal funktioner så som temperatur och ljus.

1.3 Avgränsningar

Projektet kommer att omfatta funktioner för temperaturreglering och belysningspanel. Någon montering i buss kommer inte att göras, utan en prototyp kommer utvecklas för att visa funktionen. Produkten skall anpassas för långdistansbussar, andra busstyper kommer inte vara berörda. Detta eftersom ”stadsbussar” enbart kan reglera temperaturen för förare och inte passagerare.

1.4 Precisering av frågeställningen

De frågeställningar projektet ska försöka besvara är:

- Hur viktigt är användargränssnittet för ergonomin och säkerheten?
- Vad krävs för att styra bussens funktioner via CAN?
- Hur anpassas mjukvaran för att kunna kommunicera med CAN-nätverket?

1.5 Kravspecifikation

Den kravspecifikation som tagits fram för slutprodukten är en blandning från Benteler, Volvo Bussar och författarna själv.

De krav som ställts på slutprodukten är:

- Panelen ska kunna fungera som en applikation för klimat- och belysningspanel med touchfunktionalitet.
- Styrning av funktioner via CAN.
- Programmet ska vara uppbyggt med hänsyn till ergonomi och HMI (Human Machine Interface).
- Panelen ska vara förberedd för ytterligare vidareutvecklingar.
- Panelen ska kunna ge någon typ av digital/haptisk feedback vid knapptryck.
- Knappar ska tydligt visa när de är aktiva/inaktiva.

2
Teknisk bakgrund

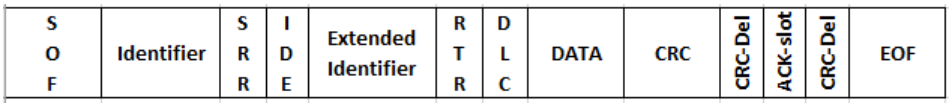
I detta kapitel sammanställs de tekniska kunskaper som använts för att genomföra detta arbete. Informationen har samlats in från datablad, tidigare examensarbeten inom området och e-böcker.

2.1 CAN (Controller Area Network)

CAN är ett fältbussystem främst anpassat för fordonsindustrin och har som uppgift att underlätta informationsbytet mellan alla enheter och minska kabelanvändningen i fordon. CAN är konstruerat så att många enheter (s.k. noder) har möjlighet att skicka och ta emot information på ett säkert och med hänsyn till prioriteringen förutsägbart sätt.
CAN utvecklades 1983 av Bosch och har på senare år blivit ett standardiserat kommunikationsnät inom fordon. Genom att använda denna typ av kommunikationsnät kan antalet kablage mellan delsystemen i fordon minskas, vilket resulterar i lägre kostnader. CAN ger också möjligheter för vidareutveckling av fler avancerade funktioner i fordonet, t.ex vid reglering av avståndet till framförvarande fordon vid användning av farthållare [1].

2.1.1 Meddelandets uppbyggnad

CAN-bus är en seriell fältbuss där all kommunikation sker i form av datapaket bestående av upp till 127 bitar. Varje meddelande har en struktur enligt figur och beskrivning nedan [2]:



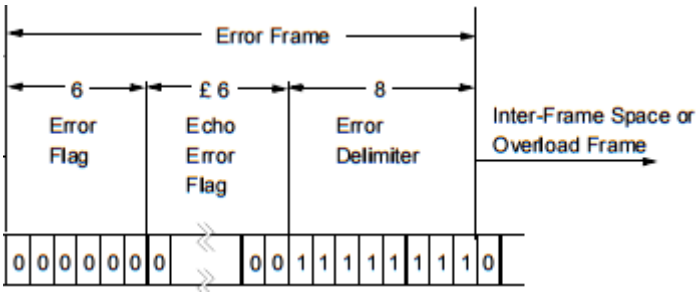
Figur 2.1: *Uppbyggnad av meddelandet*

- **SOF** (Start Of Frame): Startbiten i ett meddelande som används för att synkronisera alla noder.
- **Identifier** (ID): Anger vilken prioritet meddelandet har och fungerar samtidigt som adress till den eller de noder som är mottagare av ett visst meddelande. Om ID innehåller fler än 11 bitar används dessutom extended identifier. Lägre ID innebär högre prioritet.
- **SRR** (Substitute Remote Request): Alltid etta (1).

- **IDE** (Identifier Extenision): Anger om meddelandet är standard- eller extended frame. IDE måste vara dominant för att meddelandet ska vara en standard frame. En dominant bit betyder att biten är noll(0).
- **Extended identifier**: Används om ID innehåller 29 bitar.
- **RTR** (Remote Transmission Request): Används för att skicka en förfrågan om data från viss nod.
- **DLC** (Data Length Code): Anger hur många bytes datafältet innehåller.
- **DATA**: Innehåller 8 till 64 bitar (0-8 bytes).
- **CRC**: (Cyclic Redundancy Check): Checksumma för kontroll av att överföringen av alla bitar har skett utan fel.
- **CRC-Del**: Alltid etta (1).
- **Ack-Slot**: Noderna skickar en etta(1) till denna bit då ett meddelande tas emot korrekt. Om noderna uppfattar ett meddelande fel nollställs biten av mottagaren som sedan skickar en ”Error Frame” (se avsnitt 2.1.2).
- **Ack-Del**: Alltid etta (1).
- **EOF** (End-Of-Frame): Indikerar slutet av CAN-meddelandet.

2.1.2 Error Frame

En Error Frame genereras då noderna upptäcker ett fel i ett meddelande, som består av sex nollor(0) i rad. Error Frame skickas iväg så fort meddelandet har slutat sända. I figur nedan illustreras en Error Frame:



Figur 2.2: *Illustration av Error Frame* [2]

2.1.3 Sändning/mottagande av meddelande

Ett CAN-nätverk består av ett antal noder som är anslutna till den gemensamma fältbussen. Varje nod har ett register för mottagen data och för den data som ska sändas. För att styra mottagningen finns ett antal register som anger vilka identifierare som noden ska ta emot data från. Dessa register ska också ange vilka identifierare som noden ska ut på fältbussen. Identifieraren har i varje meddelande två uppgifter; dels ange meddelandets prioritet, samt adress till mottagande nod/noder.

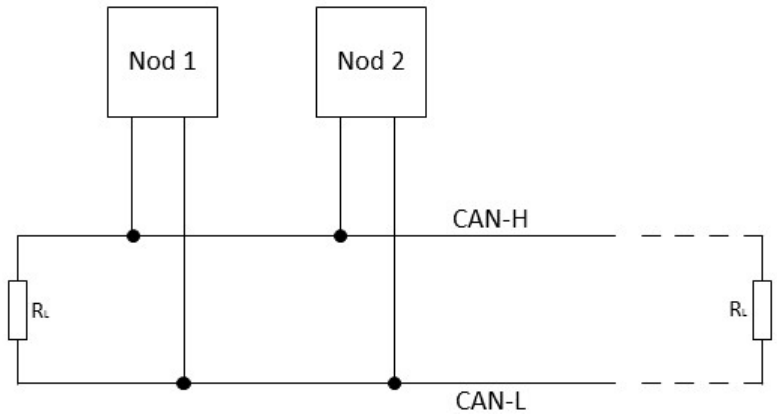
Om två eller fler noder börjar sända samtidigt kommer meddelandet med högst prioritet att vinna genom en selektion, där identifierarna sänds parallellt bit för bit. Efter det att identifierarna är sända finns det alltid bara en sändande nod kvar

som fortsätter att sända resten av meddelandet. De noder som tvingats sluta sända kommer att försöka sända på nytt när fältbussen blir ledig nästa gång. Förfarandet återupprepas tills alla noder har fått sända sitt meddelande.

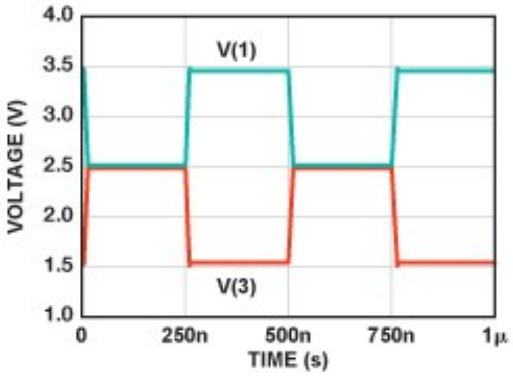
Meddelanden med hög prioritet kan till exempel vara airbags eller bromsljus. Om det finns väldigt många noder anslutna till ett system kan filter och masker användas för att filtrera bort en del av de oönskade meddelandena.

Ponera till exempel att en bilförare vill sätta på helljuset på sin bil. Då skickas ett meddelande med information om att helljuset ska aktiveras och läggs ut på nätverket. Enheten som styr helljuset läser meddelandet och aktiverar helljuset. Ska helljuset stängas av skickas ett nytt meddelande som säger att helljus ska inaktiveras.

I figur 2.3 illustreras hur noder är kopplade i ett CAN-nätverk. Nätverket har i varsin ända en last (RL = 120 Ohm) för att motverka signalreflektioner. Tolkning av meddelande i nätverket sker genom avläsning av potentialen mellan CAN-H och CAN-L. CAN-H och CAN-L har i recessivt tillstånd (när inget meddelande skickas) potentialen 2.5 V. Vid dominant tillstånd (när ett meddelande skickas) ökar CAN-H till 3.5 V och CAN-L minskas till 1.5 V (se figur 2.4) [3].



Figur 2.3: *Illustration av noder i ett CAN-nätverk*



Figur 2.4: *CAN-H och CAN-L i dominant och recessivt tillstånd* [4]

2.2 HMI, Human Machine Interface

HMI är en benämning på alla gränssnitt som används för samverkan mellan människa och maskin. HMI tar hänsyn till operatörer i industrier, användning av smartphones och datorer. Det finns mängder av mjukvara och hårdvara som kan åstadkomma detta, vissa mer generella än andra. I och med att mönstret för paneler och symboler ska kännas återkommande ändrar sig inte HMI-standarden avsevärt med åren. Detta för att personer som förflyttar sig mellan arbetsplatser och system ska känna igen utseendet [5].

2.3 Standarder

2.3.1 UNECE (The United Nations Economic Commission for Europe)

UNECE är den ekonomiska kommission för Europa som arbetar med följande huvudområden:

- miljö.
- tekniskt samarbete.
- ekonomiskt samarbete.
- transport.
- handel.

Totalt finns nio områden, där transportområdet är relevant för detta projekt.

För att displayen ska fungera som ett interface med knappar vars symboler ska representera en funktion i en buss, bör symbolerna överstämma med befintlig standard som operatören är van vid att se. Att använda sig av nyskapta symboler kan upplevas som förvirring och istället används denna standard för att minska missförstånd vid användning [6].

2.3.2 Symbolplacering

Placeringen av symboler på en aktiv display bör ske på ett organiserat och genomtänkt sätt. I ISO 16121-standarden anges hur indikatorer och manöverkontroller bör placeras. Knappar som utgör liknande funktioner bör vara grupperade då det kan upplevas som förvirring om de är utspridda. Ökning eller minskning av temperatur ska placeras på ett logiskt sätt genom att knappen för ökning bör vara placerad ovanför eller till höger om knappen för minskning av temperatur. Syftet att använda sig av en given standard är att det ligger ett stort forskningsarbete bakom informationen, vilket ger produkten en attraktiv representation och ökad användarvänlighet [7].

2.3.3 Textfonter

Texten som visas på skärmen kan använda olika typer av fonter. Då produkten bör se representativ ut för Volvo, har Volvos egna font använts.

2.4 Nuvarande reglagestyrning i bussar

Allt reglage som styr klimat och belysning i dagens moderna bussar består av hårdvaruknappar. Hårdvaruknapparna är anslutna till en styrenhet som i sin tur är ansluten i ett CAN-nätverk. Bussen har självständiga delsystem (s.k. noder) i nätverket som inriktar sig mot specifika funktioner som t.ex. innerbelysning i bussen. Sändande och mottagande av meddelande sker som tidigare beskrivet i avsnitt 2.2 [3].

Då en knapp med valfri funktion är aktiv skickas ett meddelande som säger att funktionen ska vara aktiv. Om knappen har flera lägen ändras informationen i meddelandet direkt då nytt läge blir aktivt. När en knapp inaktiveras skickas ny information ut på CAN-nätverket till den berörda styrenheten, som inaktiverar funktionen tills den får ny information om att bli aktiv igen. I figur 2.5 och figur 2.6 visas exempel på hur knapparna kan se ut för ljus- och klimatpaneler.

Figur 2.5: *Reglageknappar belysning*

Figur 2.6: *Reglageknappar klimat*

3
Metod

Inledningsvis, i väntan på svar från företaget om startdatum ska analyser göras i avseende på lämpliga programmeringsspråk. Efter val av programmeringsspråk ägnas tid till inlärning och tester samt påbörjan av projektrapport. Möten samt diskussioner görs inledningsvis tillsammans med handledare på Benteler om projektets omfattning och syfte. Genomgång av tillgänglig hårdvara som finns till förfogande kommer leda till diskussioner om hur en eventuell slutprodukt skulle kunna se ut.

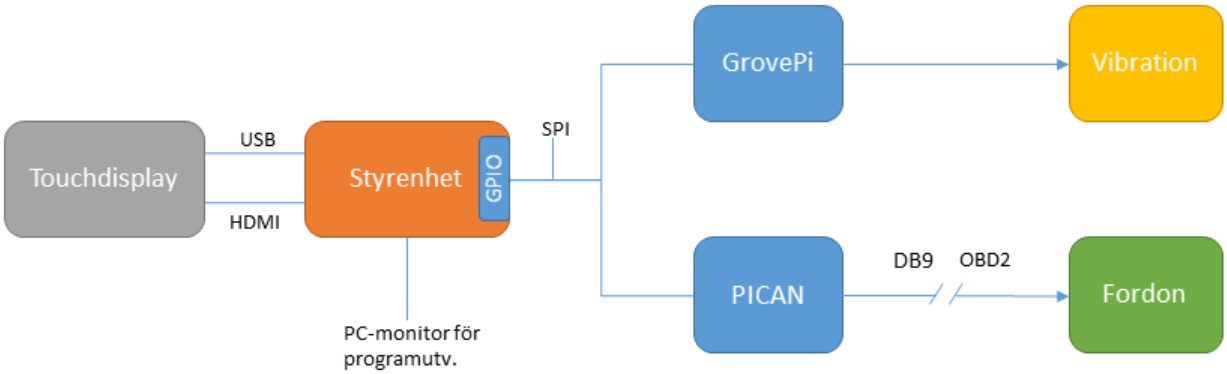
Granskning av tillgängliga hårdvaran Raspberry Pi och mjukvaran (Raspbian) görs för att få en uppfattning om dess funktionalitet och kapacitet. För att ordna upp oklarheter ordnas möten med en av företagets anställda som jobbat med utveckling av Raspberry-enheten. Tester av funktioner och grafiskt utseende kommer att ske i stor omfattning för att upptäcka idéer och skapa fler alternativ. Alla idéer ska ligga till grund för tre olika modeller som visas upp i möten med Volvo Bussar. Efter synpunkter och önskemål från Volvo Bussar sammanfattas de delar från de tre första modellerna med önskvärda ändringar och förbättringar till en slutgiltig modell.

Kompletterande information och hjälp för utveckling och förståelsen av såväl hårdvara som mjukvara hittades till största del på Internet i form av examensarbeten, forum och dokumenteringar. Även anställda på Benteler var till stor hjälp. Symboler, etiketter och knappar för displayen är skapta i Adobe Photoshop CS3.

4

Hårdvara

I figur 4.1 illustreras den hårdvara som användes under projektets gång. Sedan tidigare fanns alla hårdvarudelar tillgängliga, förutom beståndsdelarna GrovePi och vibration. I PC-datorn kan programkoden utvecklas med stöd av en utvecklingsmiljö, men går lika bra att utveckla direkt på styrenheten.



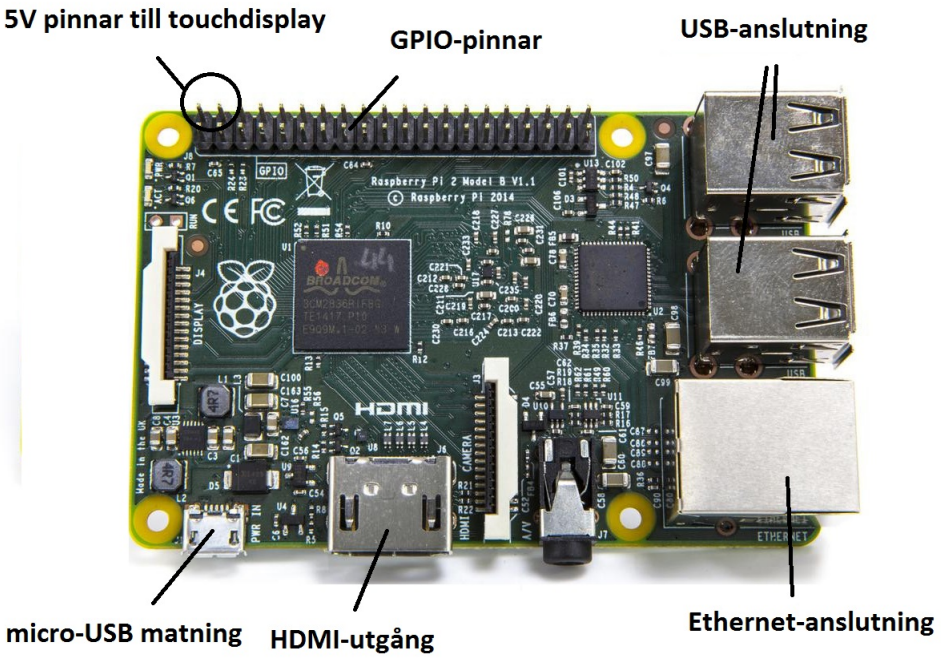
Figur 4.1: *Illustration av nätverket*

4.1 Bakgrund

För att uppfylla de krav som ställts på slutpodukten har en hårdvaruanalys gjorts. Hårdvaran ska kunna kommunicera med det kommunikationssystem som bussar använder sig av, samt ej vara allt för stor då den ska kunna monteras på fler punkter i förarhytten. Till förfogande fanns en färdigmonterad prototyp av en touchdisplay med tillhörande Raspberry Pi i en svart plastmodul. Prototypen, som konstruerades av tidigare studenter under ett examensarbete [27], hade till uppgift att underlätta övervakning av övningsförare till bussar.

4.2 Raspberry Pi

Hårdvaruanalysen som gjordes resulterade i att Raspberry Pi [8] uppfyllde kraven som bland annat innebar möjligheter för expansionskort och att den kan kommunicera via en CAN-bus.
Raspberry Pi är en dator byggt på endast ett kretskort (se figur 4.2). Dess storlek kan jämföras med ett kreditkort och finns i olika modeller. Med de krav som detta projekt ställer på processor och RAM-minne är enkortsdatorn Raspberry Pi 2 Model B relevant (se specifikationer i tabell 4.1).



Figur 4.2: *Raspberry Pi 2 model B* [9]

Specifikationer	Raspberry Pi 2 Model B
Processor	Quad-core ARM Cortex A7 900 [MHz]
Grafikprocessor	Broadcom VideoCore IV 250 [MHz]
RAM-minne	1 [GB]
Ljudutgång	3.5 [mm]
HDMI-utgång	Ja
USB	Ja (4 st)
Nätverksuttag	Ja
GPIO	40 st
Vikt	45 g
Mått	85.60 mm × 56.5 mm

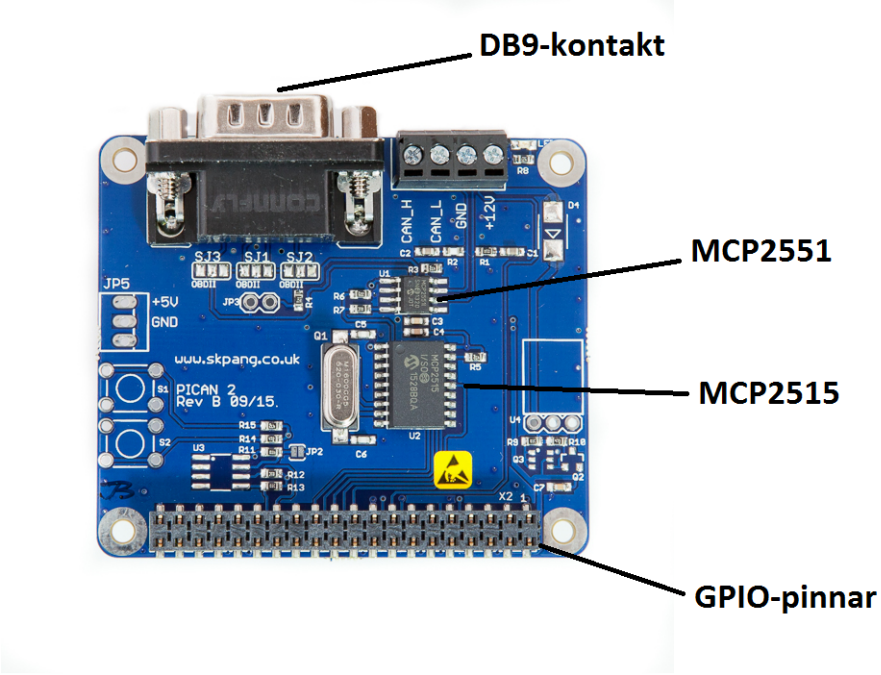
Tabell 4.1: *Specifikationer för Raspberry Pi 2 Model B* [8]

Datorn har inget inbyggt statiskt minne, istället används ett externt SD-kort för fillagring. Det betyder att datorn i sig inte heller har ett förinstallerat operativsystem. Datorn är lämpad för att använda sig av operativsystemet Linux. Fördelen med Linux är att det finns tillgängligt som öppen källkod. En anpassad Linuxdistribution som är rekommenderad för styrenheten är Raspberry Pi Foundations Raspbian som kan hämtas hem från deras hemsida och lagras på SD-kortet [10].

Fördelarna är många med denna enkortsdator, en av fördelarna är att externa moduler som ljudkort eller displayenheter enkelt kan anslutas till kortet. Det ger möjlighet till utbyggnad för fler funktioner beroende på vad som ska åstadkommas.

4.3 CAN-board PICAN

All intern kommunikation mellan enheter (noder) inom ett modernt fordon sker idag via seriella databussar. För att kunna ansluta till fordonets CAN-bus ansluts en CAN-board mellan styrenheten och CAN-bus (se figur 4.1). PICAN sköter allt som behövs för att kommunicera via CAN-bus med stöd av den mjukvara som utvecklas för Raspberry-enheten. Enheten visas i figur 4.3.



Figur 4.3: *Bild på PICAN-board för kommunikation med CAN-bus* [11]

4.3.1 MCP2515

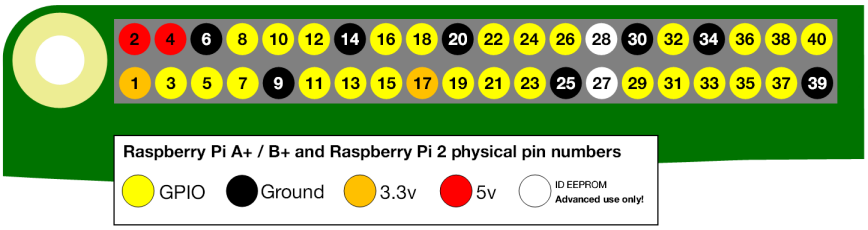
MCP2515 är en CAN-kontroller som hanterar CAN-kommunikationen. Dess huvudsakliga uppgift är att hantera det logiska gränssnittet mellan SPI och CAN. Kontrollern sitter precis under MCP2551 vid GPIO, som kan ses i figur 4.3. CAN-kontrollern har en maximal kommunikationshastighet motsvarande 1 Mb/s, som motsvarar den maximala för CAN-standarden [2].

4.3.2 MCP2551

MCP2551 är en drivkrets för anslutningen mot CAN-bus. Dess primära uppgift är att anpassa signalnivåerna, men också att skydda hårdvaran mot höga spänningstransienter och kortslutningsströmmar. På PICAN-enheten sitter microchipet mellan CAN-kontrollenheten och DB9-kontakten (se figur 4.3) [12].

4.4 Installation och strömförsörjning

Montering av PICAN-enheten på Raspberry-enheten görs via de GPIO-pinnar som sitter på Raspberry-enheten. Raspberry-enheten matas via ett micro-USB uttag (5 V DC, 1,2 A), som en mobil enhet. Val av antal GPIO-pinnar som ska användas är beroende av antalet funktioner eller externa moduler som ska anslutas. PICAN-enheten monteras från vänster då matningspinnarna sitter på GPIO-pinne 1 och 2. Valet av vilka pinnar som ska vara aktiva/inaktiva väljs sedan i mjukvaran. Illustration av GPIO-pinnar på Raspberry-enheten kan ses i figur 4.2 och figur 4.4.

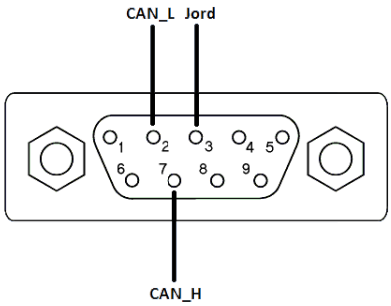


Figur 4.4: *GPIO-illustration* [13]

4.4.1 Inkoppling till CAN-bus

Vid test av kommunikation med CAN-bus ansluts en kabel med DB9-kontakt till PICAN-enheten. I andra änden av kabeln finns en OBD2-kontakt och kabeln ansluts till fordonets motsvarande OBD2-kontakt (se figur 4.6). Signaler skickas och mottages till PICAN-enheten via pinnarna på DB9-kontakten (se figur 4.5).

Exempelvis i en bil sitter service-uttaget (dvs OBD2-kontakten) under ratten bakom en lucka som man lossar på. Vid service eller testning ansluts testutrustningen via denna kontakt i bilen.



Figur 4.5: *DB9-pins* [14]

Figur 4.6: *DB9 till OBD2-kontakt* [15]

4.4.2 Skärmanslutning

Skärmen ansluts till Raspberry-enheten via USB för kommunikation med styrenheten och får matning via en matningspinne som är fastlödd på två av GPIO-pinnarna. USB-anslutningen används för att möjliggöra touchfunktionalitet. För bildanslutning används en HDMI-kabel (se figur 4.2).

Beroende på vilken version av Raspbian som ligger på Raspberry-enheten kan en del inställningar behöva göras för att få optimal anpassning. De äldre versionerna (tidigare än 2016) kräver konfiguration i operativsystemets config.txt-fil, där skärmanpassning och maximal strömförsörjning via USB-uttagen kan justeras. De nyare versionerna av Raspbian anpassar upplösningen beroende vad för skärm som ansluts, samma sak gäller strömförsörjning via USB-uttag. Fördelen med detta är att justeringar inte behöver göras kontinuerligt.

4.5 Touchdisplay

Touchsdisplayen är en LCD-skärm som visar det grafiska användargränsnittet som skapats för detta projekt (se figur 4.7 och figur 4.8). Analyser gjordes kring hur den befintliga touchdisplayen på Benteler skulle kunna användas för att nå de mål och krav som ställts. Storleken har varit viktig då man ska kunna implementera styrfunktioner utan att det känns för obekvämt för chauffören. De krav som ställts på touchdisplayen är:

- Stor yta för pekstyrningar.
- Kompatibel med Raspberry Pi.
- Touchfunktion som liknar de touchenheter som används i vardagen.
- Stark och klar ljusstyrka.
- Färgpalett där färgerna uppfattas som neutrala, då missuppfattning av färg på symboler kan innebära problem.

Figur 4.7: *Framsida display*

Figur 4.8: *Baksida display*

Touchdisplayen har följande specifikationer:

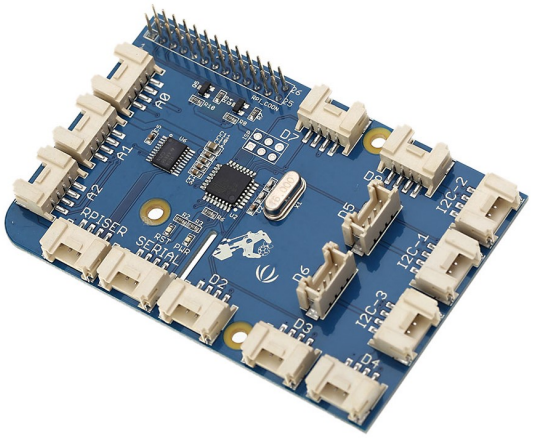
Specifikationer	LCD-skärm
Display	10” full-color a-Si TFT med IPS teknologi
Integrerad touchpanel	kapacitiv multi-touch med upp till 10 fingrar
Upplösning	1366x768 pixlar
Bildförhållande	16:9
Antal färger	16 miljoner
Dimensioner	269 [mm] x 172 [mm]
Vikt	245 gram
Matning	5 [V] DC
Effektförbrukning	2.5 [W]
Kontrastförhållande	800:1
Betraktningsvinkel	89 grader

Tabell 4.2: *Specifikationer för touchdisplayen* [16]

4.6 GrovePi

GrovePi är ett generellt I/O-kort som kan anslutas till Raspberry-enheten och är kompatibel med alla versioner av Raspberry Model B. Kortet ansluts via GPIO-pinnarna likt PICAN-enheten. Med hjälp av kortet kan externa sensorer som vibrationsmotorer, ljussensorer, temperatursensorer med flera anslutas (se figur 4.1). Det ökar möjligheterna för fler funktioner i produkten. Tack vare dess storlek och anslutningsmöjlighet sparas mycket utrymme jämfört med ett kopplingsdäck, vilket är en viktig faktor då utrymmet i monteringsboxen är begränsat.

Skaparna av kortet, Dexter Industries, har målsättningen att ersätta dagens kopplingsdäck med deras egna produkt vid produktutveckling. Deras ”plug-n-play”-enheter anslutas lätt till kortet, vilket underlättar kretsbyggandet avsevärt för olika typer av projekt. ”Plug-n-play” innebär att de externa enheterna till kortet kopplas in och kan användas omedelbart efter installation av drivrutiner [17].
I figuren nedan visas GrovePi-enheten och dess anslutningsmöjligheter:



- 7 digitala portar
- 3 analoga portar
- 3 I2C portar
- 1 seriell anslutningsport för GrovePi
- 1 seriell anslutningsport för Raspbery Pi

Figur 4.9: *GrovePi* [18]

GrovePi används för att ansluta 1-4 stycken vibrationsmotorer för montage på baksidan av touchdisplayen (se figur 4.10). Dessa används för att ge haptisk feedback (återkoppling) vid beröring. Vibrationsmotorerna ansluts med en 4-pins kabel till GrovePi-kortet. Motorerna har en vibrationshastighet på 9000 rpm [19].

Figur 4.10: *Vibrationsmotor* [20]

5
Mjukvara

5.1 Bakgrund

Mjukvaran är kärnan till användargränssnittet och funktionaliteten. Valet av programmeringsspråk Python gjordes genom att välja samma programmeringsspråk som föregående examensarbete [27]. Med tanke på tidigare erfarenheter och förkunskaper hade det varit mer lämpligt med C, men för att bygga upp ett grafiskt och touchkompatibelt användargränssnitt var Python mer lämpat för projektet. Fördelen med Python är att det för denna tillämpning finns ett mycket bra stöd för att utveckla ett grafiskt HMI i form av ett grafikpaket Tkinter. En faktor som skulle kunna spela roll är skillnaden i exekveringshastighet mellan C och Python. C kompilerar och bygger programkoden innan programmet körs, där efter exekveras programmet. Python däremot kompilerar och bygger programmet under tiden det körs, vilket påverkar exekveringshastigheten. Dock var detta ingen avgörande faktor för val av språk, eftersom exekveringshastigheten inte hade någon större inverkan [21].

5.2 Mjukvarudelar

Detta kapitlet beskriver viktiga beståndsdelar för projektets mjukvara.

5.2.1 Raspbian

Raspbian är ett operativsystem anpassat för användning av Raspberry Pi, som är en Linuxdistribution. Raspbian och är en modifierad Linuxversion utvecklad från Debian Wheezy. Tack vare att Raspbian har öppen källkod finns det en stor grupp av användare och utvecklare. Raspbian lämpar sig bra som operativsystem med funktioner som mapphantering, menyer, webbläsare mm.

Den version av Raspbian som använts i detta projekt är NOOBS (2016-09-30). Fördelen med NOOBS är att användargränssnittet är lättanvänt och mer eller mindre färdigkonfigurerat. Via startmenyn kan interna moduler aktiveras, det är till exempel här SPI-modulen aktiveras för att upprättahålla kommunikation med PICAN och dess komponenter.

Användargränssnittet i NOOBS-distributionen skiljer sig inte allt för mycket mot de operativsystem som används mest i dagsläget(Windows, IOS). Främst för att

det ska ge ett hanterbart och inte allt för komplicerat gränssnitt. Filer, mjukvarumoduler och uppdateringar installeras genom systemets terminal, vilket kan kännas ovant. Men fungerar precis på samma sätt som i t.ex Windows och IOS, bara att det där är mjukvarustyrt.

5.2.2 Python

Python är ett interpreterande språk som till stor del liknar C och C++. Det finns skillnader som stärker Python's koncept med enkelhet jämfört med andra nämnda språk; kodrader behöver inte avslutas med semikolon, indragningar framför kodrader utesluter användingen av ”måsvingar”, många färdiga moduler som kan installeras via terminal och kan börja kodas med direkt [22].

5.2.2.1 Tkinter

Tkinter är ett standardbibliotek för Python som tillåter användaren att skapa ett lämpligt användargränssnitt mellan människa och dator. Tkinter använder sig av bilder och grafiska element för att designa ett gränssnitt. Exempel är tryckknappar, textflikar, importering av bilder, placering med hjälp av koordinater m.m. [23].

5.2.2.2 Spidev

Spidev är en drivrutin som finns inbäddat i Python, som används för att styra SPI-modulen.

5.2.2.3 GrovePi

För att kunna kommunicera med sensorenheten krävs installation av mjukvara på Raspberry-enheten. Mjukvaran ger färdigkonfigurerade inställningar. Inkopplade sensorer på enheten testas genom att köra exempelprogram som medföljer. Fungerar exempelkoden kan implementering av sensoraktivitet ske i huvudprogrammet med hjälp av modulen.

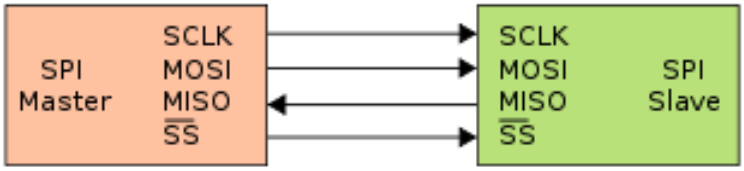
5.2.2.4 CANalyzer

Vid test av kommunikation av CAN-bus används CANalyzer för analys av datatrafik. I mjukvaran kan meddelanden sändas och tas emot.

5.2.3 SPI - Serial Peripheral Interface

SPI är ett enkelt och snabbt sätt att kommunicera med flera styrenheter samtidigt. SPI är ”full duplex”, som innebär att den kan både skicka och ta emot information samtidigt. Med hjälp av drivrutinen Spidev i Python kan kommunikation mellan Raspberry-enheten och CAN-kontrollen upprätthållas. Tekniken utgörs av en ”master” som läser och skriver till ”slaves”, i detta fall CAN-kontrollen [24]. Figur 5.1 visar grafiskt hur kommunikation sker genom fyra vägar:

- SCLK : Serial Clock (output from master)
- MOSI : Master Output, Slave Input (output from master)
- Master Input, Slave Output (output from slave)
- SS : Slave Select (active low, output from master)



Figur 5.1: *Illustration av SPI-kommunikationen* [25]

5.3 Grafisk representation och utveckling

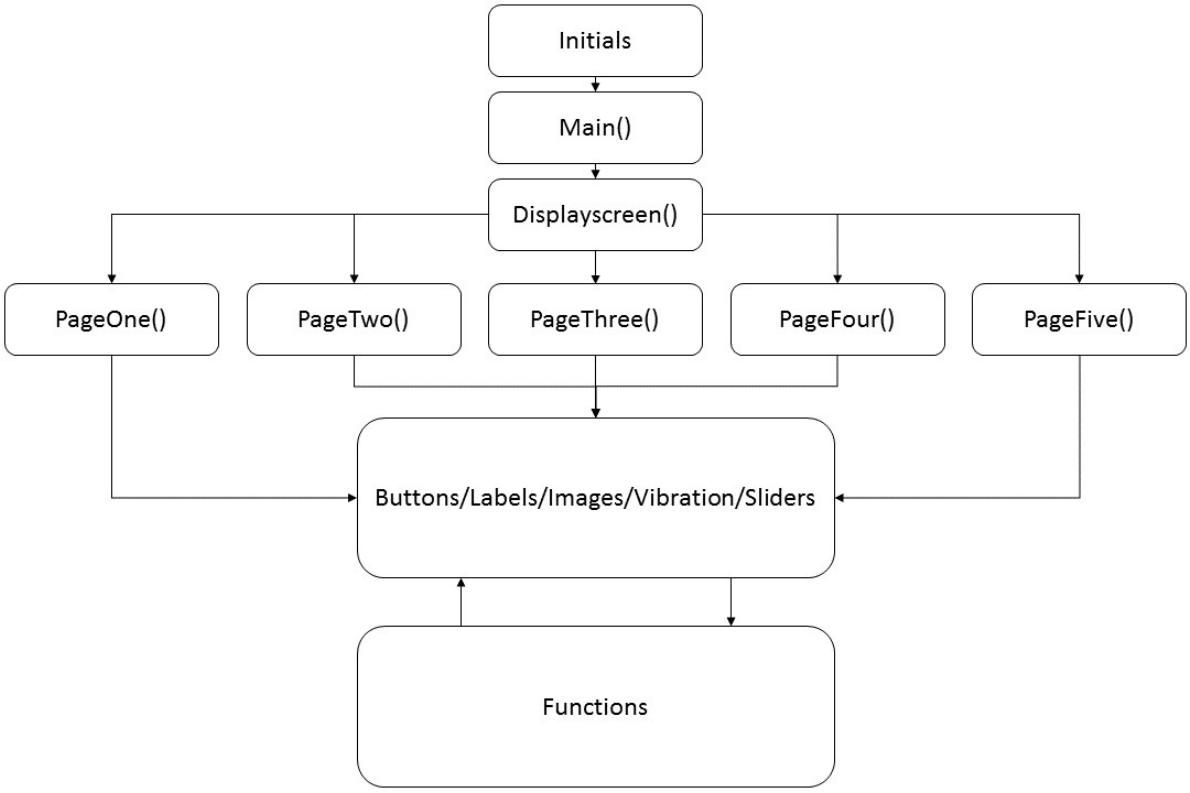
Programmet är enbart utvecklat av författarna själva. Det projektet bygger vidare på från examensarbete [27] är dess hårdvara samt CAN-kommunikation.

Genom projektets gång skissades flera olika modeller på papper om hur användargränssnittet kunde representeras på skärmen, hur knappar skulle vara placerade och hur hela ytan kunde fyllas upp.
För att kunna utnyttja den kreativa förmågan skapades symboler och etiketter i Adobe Photoshop, vilket gav fler möjligheter för designval och layout. Placeringar av användargränssnittets objekt följde ISO16121-standard [7], examensarbete [26] och input från Volvo Bussars HMI-avdelning.

Analys- och forskningsarbetet som examensarbete [26] gjorde gav mycket information om hur klimatpanelen skulle kunna se ut och ha för funktioner, då de i arbetet tog fram alternativa placeringar av knappar och etiketter, med hänsyn till ISO16121-standarden. Informationen som de genom analys- och forskning fick fram var stödjande i hela projektets gång, då djupgående information behövdes.

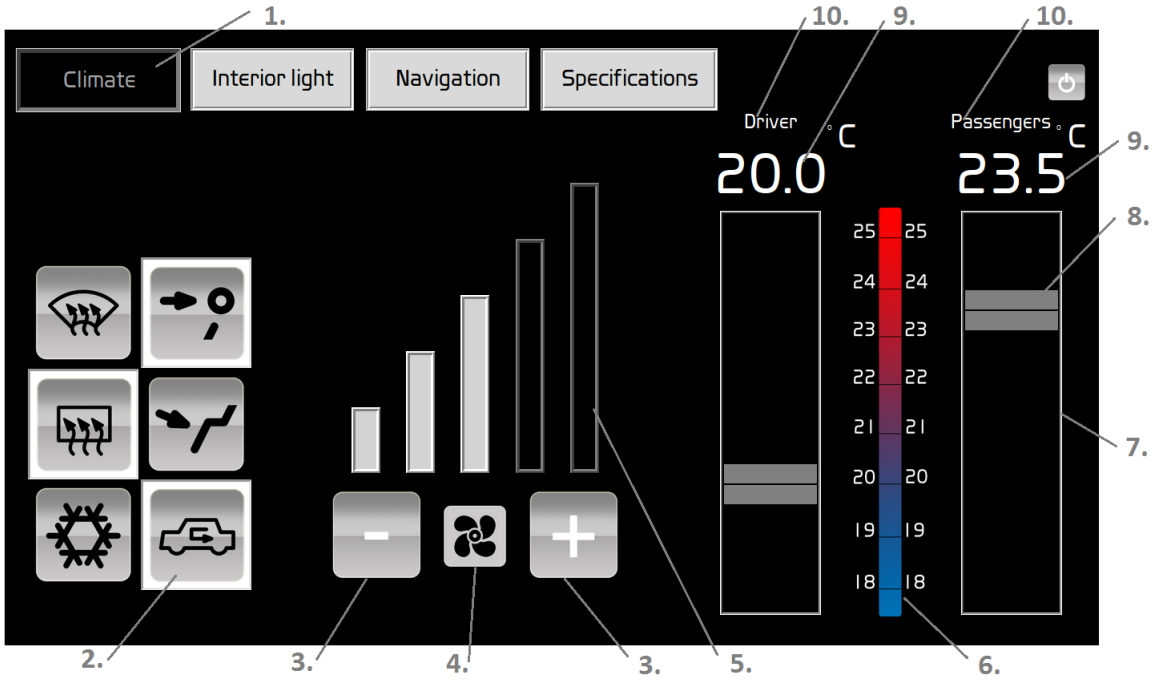
5.4 Design av programkod

Programkoden är uppdelad enligt strukturen i figur 5.2 och detta kapitel kommer beskriva programmets olika delar. Koden är uppbyggd i klasser som används för att få en bättre struktur och gör det lättare att orientera sig. I klasser kan lokala variabler och fonter deklareras, som enbart kan användas i just den klassen. All kod är hämtad från bilaga 1.



Figur 5.2: *Struktur över programkod*

Alla objekt som anges i kapitlet hänvisas till nedanstående figur:



1. Menysida
2. Klimatfunktioner
3. Ökning/minskning av fläktnivå
4. Fläktsymbol
5. Fläktnivåer
6. Temperaturgradient med siffror
7. Temperaturreglage
8. Slider
9. Aktuell temperatur i siffror
10. Temperatursektion

Figur 5.3: *Objekt i slutprodukten*

Programmet har fyra menysidor som innehåller menyknappar och tillåter användaren att välja mellan menyalternativen. Den aktuella sidans menyknapp framstår tydligt som aktiv, som efterliknar en knapp som är intryckt (objekt 1 i figur 5.3).

5.4.1 Initials, Main och Displayscreen

Initials importerar följande bibliotek och drivrutiner till programmet:

- Tkinter (se kapitel 5.2.2.1)
- tkFont (Ger möjlighet att välja font)
- PIL (Python Imaging Library, bild och grafik-drivrutin)
- Time (Gör det möjligt att fördröja funktioner)
- Grovepi (GPIO-hantering för GrovePi)

Storleken på programmets ram, alltså vad som visas när programmet körs, bestäms i Main. Önskas användning av samma variabler eller fonter för flera

klasser deklareras dessa i Main och blir globala. Main startar också klassen Displayscreen vid exekvering. Displayscreen är en klass som hanterar alla menysidor i programmet.

5.4.2 PageOne

PageOne är menysidan för klimatpanel. Användaren kan ändra temperatur, öka/minska fläktstyrkan eller använda tillgängliga klimatfunktioner.

Fonter och variabler
Lokala fonter deklareras på samma sätt som de globala, med undantaget att de endast kan användas i den klassen. Variabler i Tkinter anges som antingen IntVar, DoubleVar, StringVar eller BooleVar.

```
###----------------------------Variables-----------------------------###
  # IntVar: Variable as integer
  # DoubleVar: Same as IntVar, but with decimal
  # StringVar: Variable that updates texts
  # A DoubleVar is needed so that the GUI updates whenever the variable changes

        self.TempPassengers = DoubleVar()
        self.TempPassengers.set(20.0)    # Initial value

        self.FanLevel = IntVar()
        self.FanLevel.set(0)

        self.MaxFanLevel = 5
        self.MinFanLevel = 0

        self.FanPosTop = StringVar()
        self.FanPosTop.set("NEUTRAL")
###--------------------------Variables end-----------------------###
```

Bilder
Bilder importeras och skalas efter önskad storlek.

```
###----------------------------Images-----------------------------###
        self.AddFanImage = Image.open("Add.png")
        self.AddFanImage = self.AddFanImage.resize((120, 120), Image.ANTIALIAS)
        self.AddFanImage = ImageTk.PhotoImage(self.AddFanImage)
###--------------------------Images end-----------------------###
```

Klimatknappar och etiketter
Knappar skalas och placeras på önskad plats. När en knapp aktiverats kallar den på en funktion som utför önskad händelse. Detta sker med alternativet *command* (se programkod nedan), som väljer en funktion som ska utföras. Knappar har möjlighet att ändras grafiskt så som bakgrundsfärg, 3D-style, aktiveringstillstånd m.m. När användaren trycker på en knapp konfigurerats samtidigt knappens bakgrundsfärg, ramen runt knappen, i användargränsittet för att tydlig indikera att knappen aktiverats (objekt 2 i figur 5.3).

```
###--------------------------Buttons-----------------------------###
        # Command:  Run a function when pressed
        # Relief:  3D button-style
        # relx:  X-coordinates
        # rely:  Y-coordinates
        # anchor:  set a position (right, left, center etc)
        # state: ON/OFF setting (NORMAL/DISABLED)

        self.AddFanButton = Button(self, image=self.AddFanImage, bg="black",
↪   highlightbackground='black',activebackground="black", relief=FLAT)
        self.AddFanButton.place(relx=.52, rely=0.82, anchor="c")
        self.AddFanButton.config(width=120, height=120)
        self.AddFanButton.bind('<ButtonPress-1>',self.AddFanButtonPress)
        self.AddFanButton.bind('<ButtonRelease-1>',self.AddFanButtonRelease)
```

Etiketter framstår som bild- eller textformat och kan på samma sätt som knappar ändras grafiskt efter behov. Detta används till symboler, texter och den inställda temperaturen i siffror (objekt 4,5,6,9,10 i figur 5.3).

Temperaturreglage

Temperaturreglaget använder en slider för att ställa in önskad temperatur på en angiven skala. Önskad temperatur fås även vid tryck på önskad temperatur på temperaturreglaget. Slidern förflyttas då till önskad heltal och temperaturen kan därefter regleras som tidigare (objekt 7,8 i figur 5.3). Se utdrag av kod nedan hur känsligheten ställs in på reglagen.

```
# Defines between which coordinates the user clicked on the scale, and sets the
↪  temperature.
   def SetScalePassengers(self):
     self.Vibration()
     # Minimum distance from top=227
     # Maximum distance from top=728
     y_coord = app.winfo_pointery() - app.winfo_rooty() #Mouse x-coordinates
↪  depending on the size of the screen
     if y_coord >= 227 and y_coord < 292:
      self.TempScalePassengers.set(18.0)
     elif y_coord >=292 and y_coord < 354:
      self.TempScalePassengers.set(19.0)
     elif y_coord >=354 and y_coord < 416:
      self.TempScalePassengers.set(20.0)
     elif y_coord >=416 and y_coord < 478:
      self.TempScalePassengers.set(21.0)
     elif y_coord >=478 and y_coord < 540:
      self.TempScalePassengers.set(22.0)
     elif y_coord >=540 and y_coord < 602:
      self.TempScalePassengers.set(23.0)
     elif y_coord >=602 and y_coord < 664:
      self.TempScalePassengers.set(24.0)
     elif y_coord >=664 and y_coord < 728:
      self.TempScalePassengers.set(25.0)

     self.TempScalePassengers.config(state=DISABLED) # Stops the slider from
↪  increasing/decreasing constantly
```

Fläktnivå

Fläktnivån, även kallad fläktstyrkan, kan ställas in med knapparna ”-” eller ”+”. För att snabbt maximera eller minimera fläktnivån kan vald knapp hållas nedtryckt en kortare tid (objekt 3 i figur 5.3).
Aktuell nivå visas med etiketter som släcks och tänds beroende på önskad effekt (objekt 5 i figur 5.3).

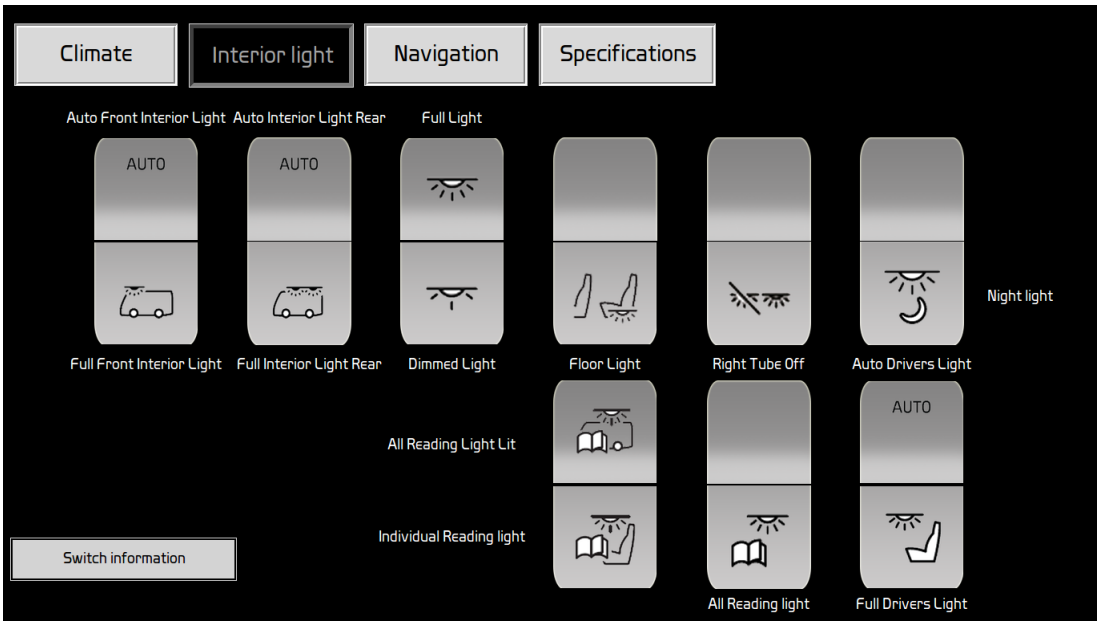
```
###---------------------------Fan levels----------------------------###
    # Maximize fanlevel if button is pressed more then 400ms
    def AddFanButtonPress(self,event):
                    self.ButtonHolded.set(app.after(400, self.SetFanToMax))

    # Maximize fanlevel
    def SetFanToMax(self):
                    self.Vibration()
                    self.FanLevel.set(self.MaxFanLevel)
                    self.Add_Fan()

    # Cancel maximum level if button is realeased within 400ms, and increase
↪    fanlevel if its not already maximized
    def AddFanButtonRelease(self, event):
                    app.after_cancel(self.ButtonHolded.get())
                    if self.FanLevel.get() >= self.MaxFanLevel:
                        pass
                else:
                        self.Add_Fan()

    # Increases fanlevel, or passing if its already maximized
    def Add_Fan(self):
                    self.Vibration()
                    if self.FanLevel.get() >= self.MaxFanLevel:
                            self.Fanbar1Label.config(bg="lightgrey")
                            self.Fanbar2Label.config(bg="lightgrey")
                            self.Fanbar3Label.config(bg="lightgrey")
                            self.Fanbar4Label.config(bg="lightgrey")
                            self.Fanbar5Label.config(bg="lightgrey")
                            pass
                    else:
                            self.FanLevel.set(self.FanLevel.get()+1)
                            if self.FanLevel.get() == 1:
                                    self.Fanbar1Label.config(bg="lightgrey")
                            elif self.FanLevel.get() == 2:
                                    self.Fanbar2Label.config(bg="lightgrey")
                            elif self.FanLevel.get() == 3:
                                    self.Fanbar3Label.config(bg="lightgrey")
                            elif self.FanLevel.get() == 4:
                                    self.Fanbar4Label.config(bg="lightgrey")
                            elif self.FanLevel.get() == 5:
                                    self.Fanbar5Label.config(bg="lightgrey")
###-------------------------Fan levels end-------------------------###
```

5.4.3 PageTwo

I denna menysida har användaren möjlighet att ändra bussbelysningen. Menyknappar, fonter, variabler, bilder och ljusknappar har deklarerats på samma sätt som i PageOne, med annorlunda utseende.

En knapp, ”Switchinformation”, har skapats för att upplysa användaren om vad varje knapp har för innebörd. Vid knapptryck visas etiketter med knappinformation, som släcks efter en stund (se figur 5.4). Detta för att minska missförstånd och tiden det tar att hitta rätt ljusknapp.

```
###------------------------------Buttons------------------------------###
                self.SwitchInformation = Button(self, text="Switch
 ↪  information", bg="lightgrey", relief="solid",
 ↪  command=self.SwitchInformationPress, width=23, height=2,
 ↪  font=self.SwitchInformationFont, bd=2)
                self.SwitchInformation.place(relx=.11, rely=0.9, anchor="c")
###----------------------------Buttons end----------------------------###
###----------------------------Functions-----------------------------###
        # Change foreground color of each switchlabel to white when pressed.
 ↪  Active in 5 seconds
        def SwitchInformationPress(self):
                self.FullLightLabel.config(fg="white")
        # Timer function for switchinformation released
                app.after(5000, self.SwitchInformationOff)
        # Change foregroundcolor of each switchlabel to black
        def SwitchInformationOff(self):
                self.FullLightLabel.config(fg="black")
###----------------------------Functions end-------------------------###
```



Figur 5.4: *Belysningspanel för slutprodukt med aktiverade knappar*

5.4.4 PageThree, PageFour och PageFive

Inlagda menysidor som illusterar framtida vidareutveckling.

5.4.5 CANalyzer

Med hjälp av CANalyser anslöts PICAN-enhetens DB9-kontakt med ett CANalyzer-instrument för att analysera test av kommunikation på CAN-bus. Via CANalyzer kunde meddelanden studeras från en log-ruta. Sändning/mottagande av meddelande gjordes med programkod från examensarbete [27] för att förstå hur kommunikationen på CAN-nätverket fungerar. Kommunikationen till olika noder via CAN-bus från styrenheten utvecklas med stöd av ett befintligt funktionspaket. Utöver lämplig initiering av de aktuella kommunikationsparametrarna finns det tillgång till funktioner för att sända och ta emot data.

En sändning genomförs genom att initiera register i CAN-modulen med önskad identifierare (ID), motsvarande adresserad nod och prioritet. Önskad data till noden (DATA) som kan bestå av upp till 8 byte. Därefter skickas en sändinstruktion till CAN-modulen. CAN-modulen tar över ansvaret och kommer att genomföra sändningen så snart som möjligt med hänsyn till aktivitet på CAN-bus och meddelandets prioritet. Nedan visas ett utdrag av kod för en sändning över CAN-bus till viss nod hämtat från examensarbete [27].

```
#Send FRAME
def send_FRAME(ID ,DATA):
        WRITE_SPI          = 0b00000010
        READ_SPI           = 0b00000011
        #Choose buffer
        TXBnCTRL_address = 0x30
        for x in range(0,3):
                test = spi.xfer2([READ_SPI, TXBnCTRL_address, 0x00])
                if (test[2] & 0b00001000) == 0:
                        break
                else:
                        TXBnCTRL_address += 0x10
        #Fill identifyer and data register
        LISTofRegistervalues= [WRITE_SPI, TXBnCTRL_address+1,
↪   IDentifyer[ID][0], IDentifyer[ID][1], IDentifyer[ID][2], IDentifyer[ID][3],
↪   IDentifyer[ID][4]]

        #Add data to list to send via SPI
        if IDentifyer[ID][4] >= 1:        #DATA is a list of bytes
                LISTofRegistervalues.extend(DATA_to_bytes(DATA,
↪   IDentifyer[ID][4])) #IDentifyer[ID][4] = DATALENGTH in bytes
        #Load the FRAME in the selected buffer
        spi.xfer2(LISTofRegistervalues)

        #Reguest to send (RTS)
        if TXBnCTRL_address == 0x30:        #Transmitt buffer 0
                spi.xfer2([0b10000001])
```

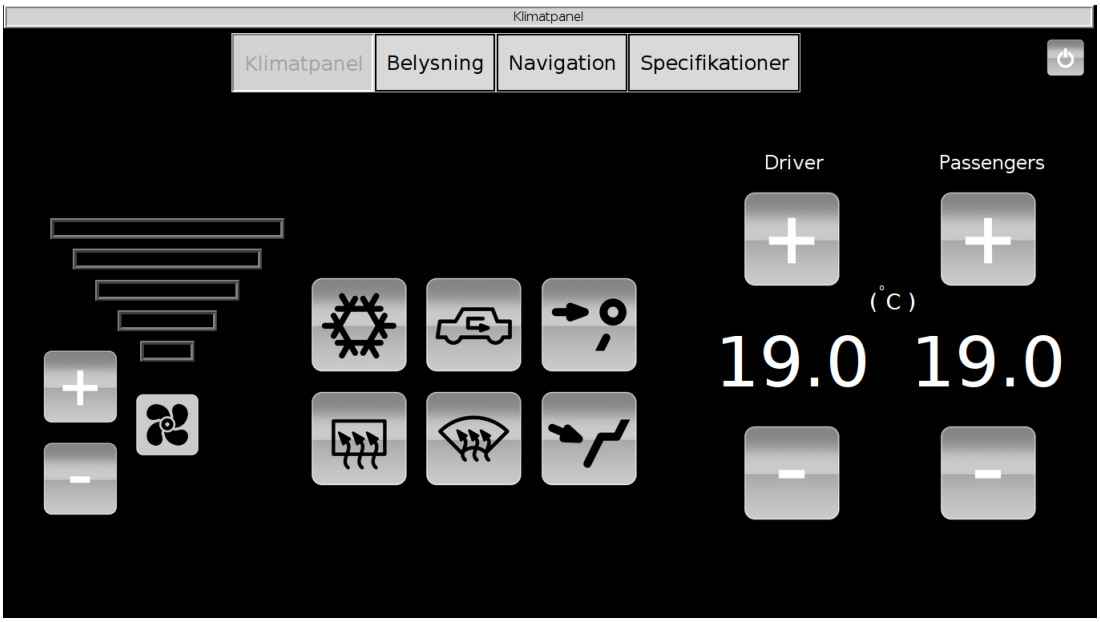
```
        elif TXBnCTRL_address == 0x40:          #Transmitt buffer 1
                spi.xfer2([0b10000010])
        elif TXBnCTRL_address == 0x50:          #Transmitt buffer 2
                spi.xfer2([0b10000100])
```

6
Slutprodukt

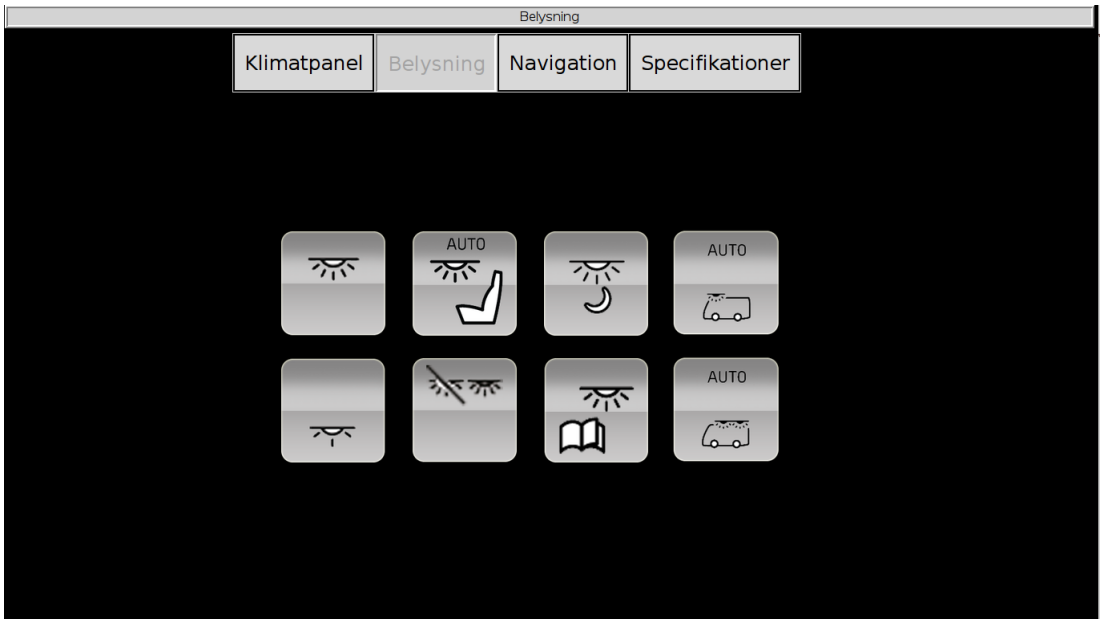
Skapandet av tre modeller låg till grund för slutprodukten.
Modellerna visade prov på olika typer av reglage, knapplaceringar och design för klimat och belysningspanel, men också möjligheterna för ytterligare funktionalitet i form av navigation och teknisk specifikation. Modellerna presenterades för Volvo Bussar utifrån författarnas omdöme i form av styrkor. Modellerna hade alla olika koncept med en röd tråd. Bussbelysningen i samtliga modeller skiljer sig markant, detta eftersom första mötet med Volvo Bussar skulle fokusera på klimatpanelen.

Modell 1
Konceptet är att efterlikna befintliga klimatpaneler med enbart knappar som reglage (se figur 6.1). I figur 6.2 visas knapparna till bussbelysningen för modell 1 och modell 2.

Styrkor:
- Stilren design.
- Enbart reglage i form av knappar.
- Enkel att förstå.
- Tydlig gruppering.



Figur 6.1: *Klimatpanel modell 1*



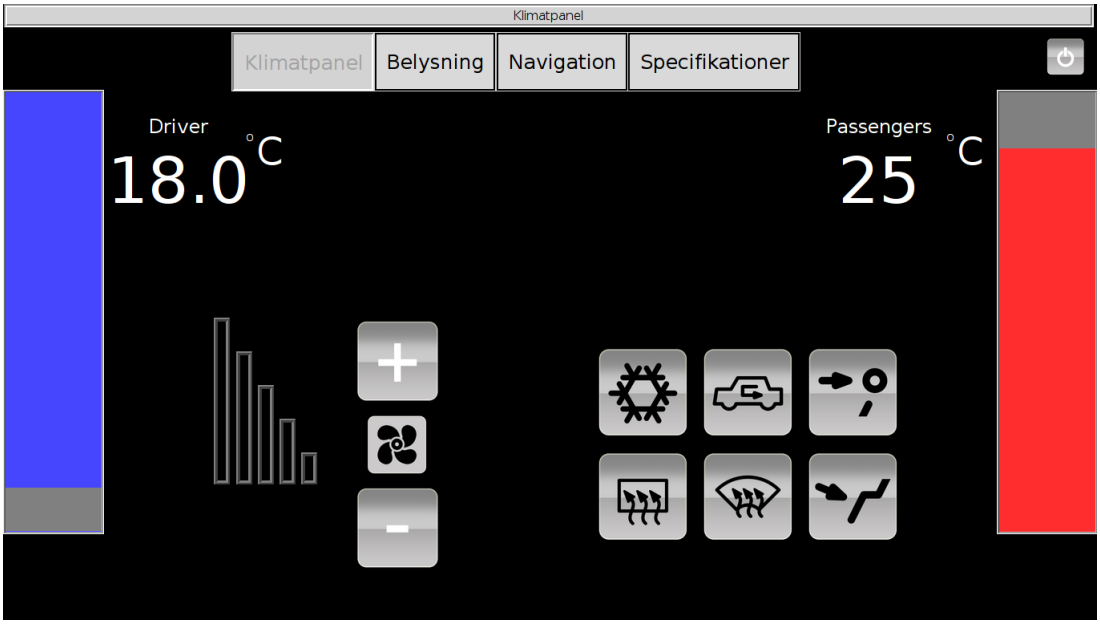
Figur 6.2: *Belysning för modell 1 och 2*

Modell 2

I modell 2 bytes knapparna för inställning av temperatur ut mot temperaturreglage med slider. Syftet var att efterlikna klimatpaneler i Volvos 90-serier för bilar. Fokus låg också på att effektivisera utrymmet, genom att placera fläktnivåknapparna på höjd och temperaturreglagen längs sidorna (se figur 6.3).

Styrkor:

- Modern design i form av reglage med sliders.
- Effektiv användning av yta, plats för fler funktioner.
- Gradient feedback vid ändring av temperatur med sliders.

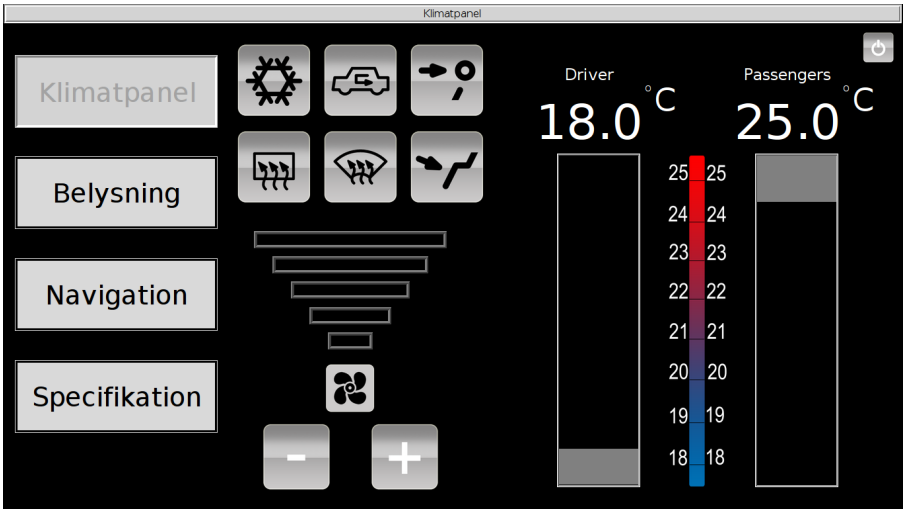


Figur 6.3: *Klimatpanel modell 2*

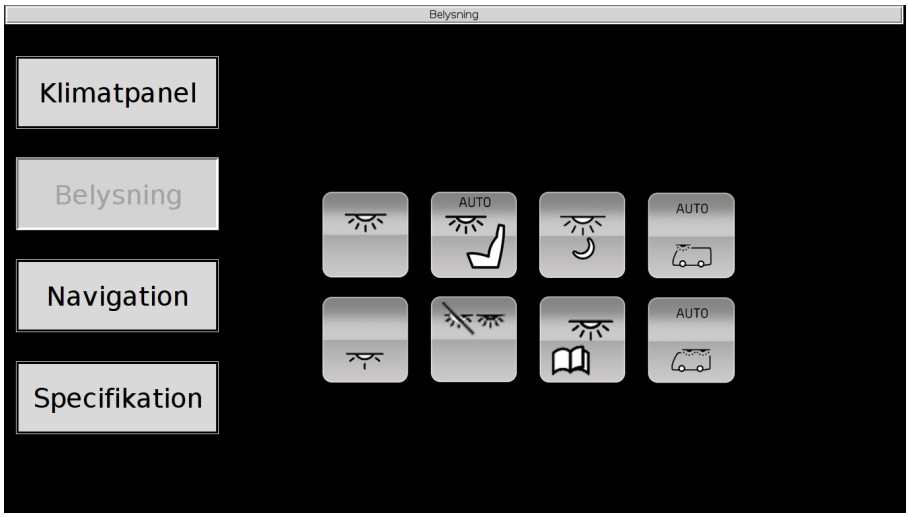
Modell 3
Genom att förstora alla tillgängliga knappar ska det vara lättare för föraren att se och välja rätt inställning.
Temperaturreglagen från modell 2 placerades bredvid varandra med en temperaturgradient i mitten. Tanken är att ge en tydligare uppfattning om den inställda temperaturen och det minsta och högst inställbara temperaturen (se figur 6.4). I figur 6.5 visas bussbelysningen, som endast har större menyknappar jämfört med de tidigare modellerna.

Styrkor:

- Modern design i form av reglage med sliders.
- Dubbel feedback av temperatur, både med siffror och temperaturgradient med sifferskala.
- Stora menyknappar.



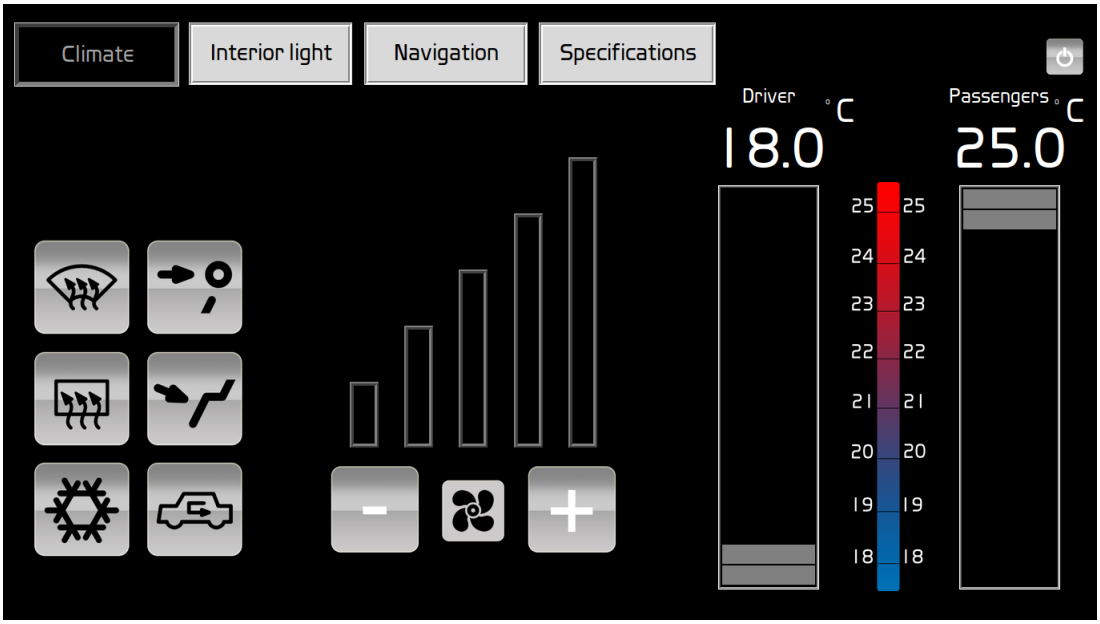
Figur 6.4: *Klimatpanel modell 3*



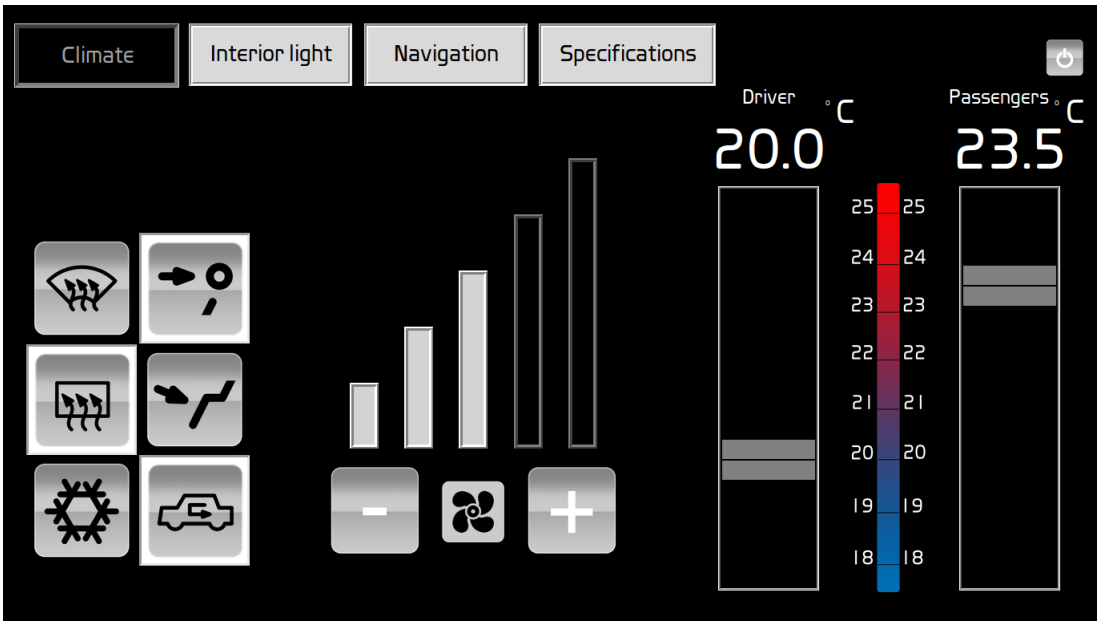
Figur 6.5: *Belysning modell 3*

Slutgiltlig modell

Slutprodukten blev efter möte med Volvo Bussar en blandning av de tre modeller. Argumenten från Volvo Bussar var att modell 1 var lätt att orientera sig i, modell 2 hade bra effektivisering av utrymme och modell 3 var modern och futuristisk med temperaturreglage och temperaturgradient. Önskemål och ändringar för den slutgiltiga modellen togs emot, resultatet kan ses i figur 6.6 till och med figur 6.12:

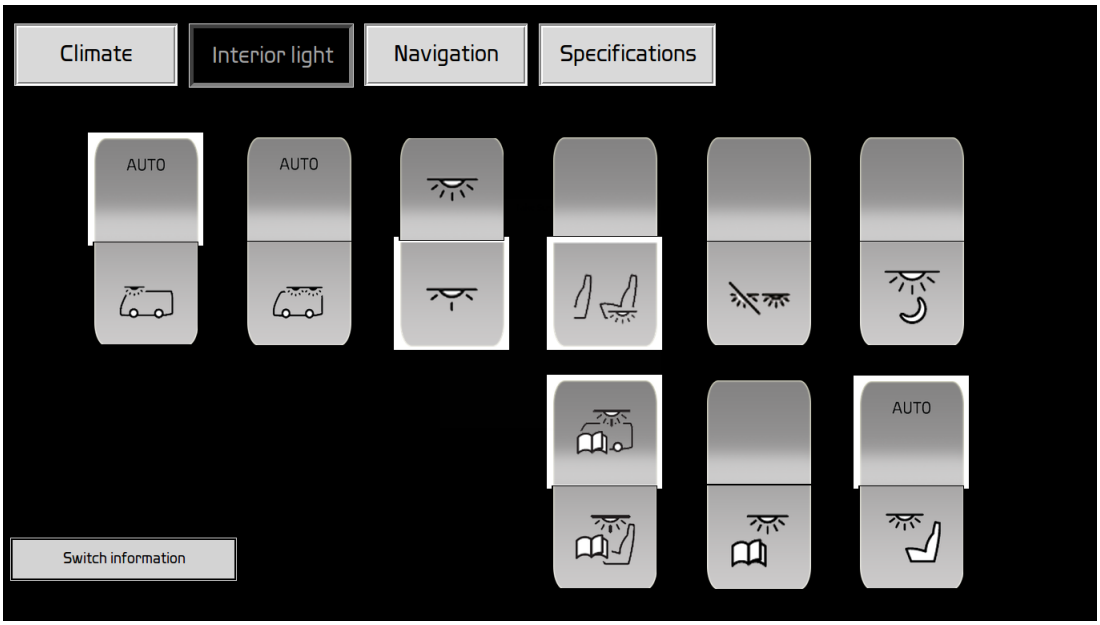


Figur 6.6: *Klimatpanel för slutprodukt*

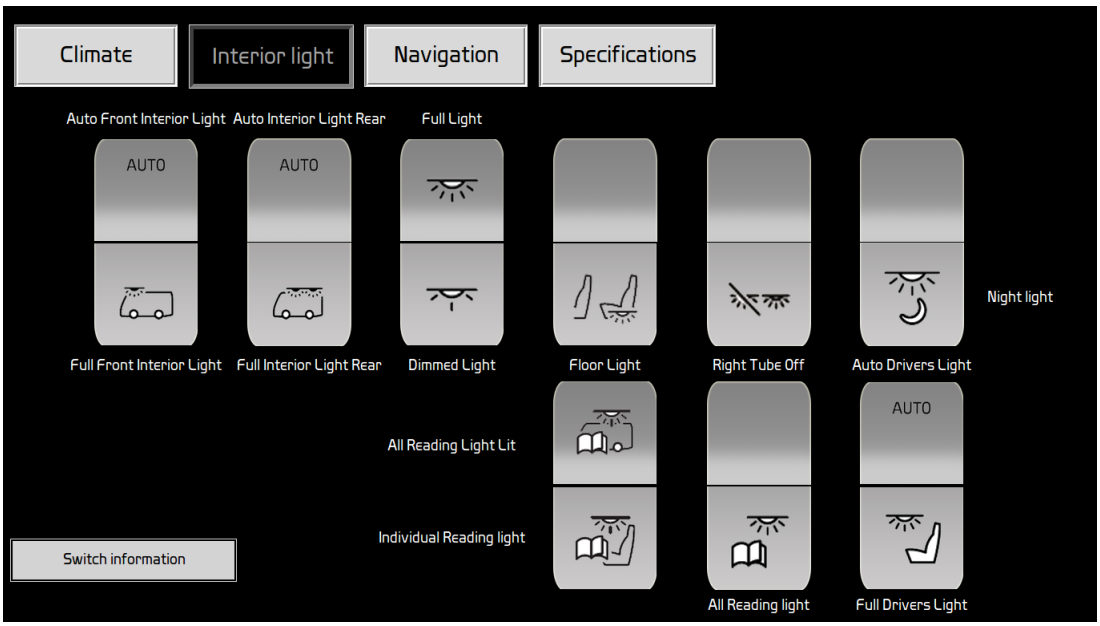


Figur 6.7: *Klimatpanel för slutprodukt med aktiverade knappar och etiketter*

Belysningsknapparna som kan ses i figur 6.2 och figur 6.5 har blivit designade till switchar, istället för knappar. Anledningen är att switcharna ska efterlikna hårdvaruswitcharna som i dagsläget återfinns i bussar.

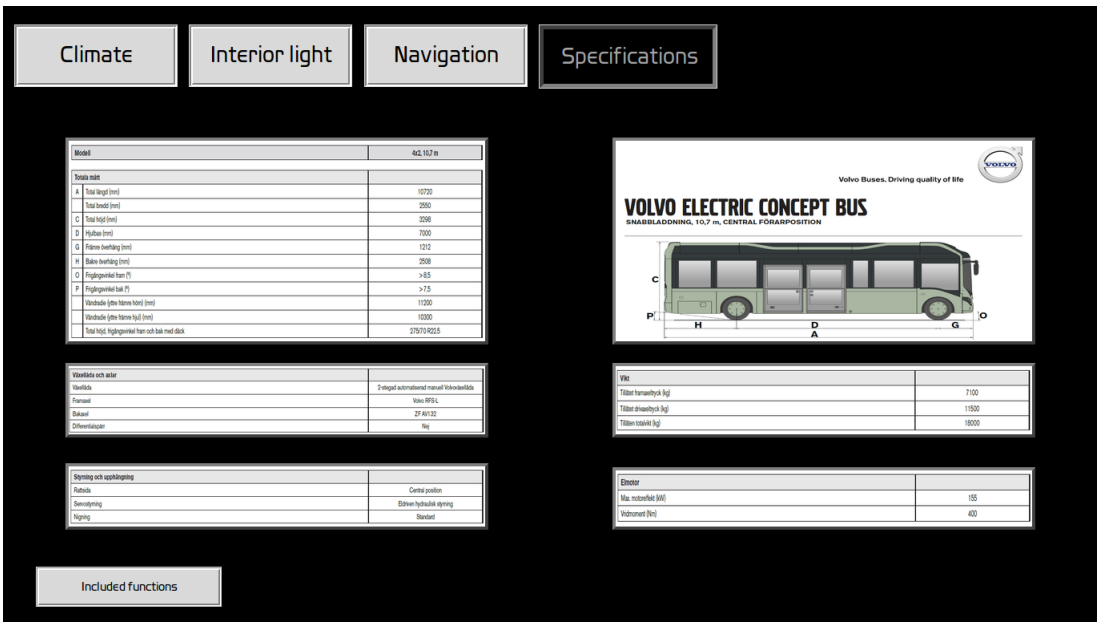


Figur 6.8: *Belysningspanel för slutprodukt med aktiverade knappar*



Figur 6.9: *Belysningspanel för slutprodukt med knappen "switch information" aktiverad*

Figur 6.10: *Navigation för slutprodukt (endast en bild som illustration)*



Figur 6.11: *Specifikationer för slutprodukt*

Figur 6.12: *Slutprodukt i drift*

Utöver att funktioner och reglage från de tre modellerna blivit ihopslagna till ett slutligt användargränssnitt, har funktionsknapparna fått ett tydligare utseende när de är aktiverade. Menyknapparna flyttades åt vänster så föraren har knappar vänsterplacerade i och med att fordonet är vänsterstyrt och vara närmare till hands.

Etiketten som är placerad i överkant, som kan ses i de tre första modellerna, visar vilken sida som är i fokus. Denna blev borttagen i slutliga modellen då det ansågs vara dubbel information.

7
Resultat

7.1 Frågeställningar

Hur viktigt är användargränssnittet för ergonomin och säkerheten?
För att produkten ska kännas användarvänlig är användargränssnittet en viktig faktor. Svårigheter att göra inställningar leder till fler rörelser och gester som inte är bra ur en ergonomiskt synpunkt. Ett otydligt och svårhanterligt användargränsitt tar uppmärksamheten från vägbanan och ökar risken för olycka. Genom möten med Volvo Bussar och från analyser och resultat från examensarbete [26] har användargränssnittet utvecklats för att representera ergonomisk design och funktionalitet. Resultatet av detta är att slutproduktens gränssnitt är anpassat för chaufförer.

Hur går man tillväga för att styra bussens funktioner via CAN?
Analyseringar av datablad till CAN-kontrollenheten [2] och sändare [12] gjordes för att förstå hur signaler skickas och mottages via PICAN. Hur signaler i CAN-nätverket skickas och mottages redovisas i avsnitt 2.2.

Hur anpassar man sin mjukvara för att kunna kommunicera med CAN-nätverket?
Arbete gjordes för att försöka uppnå sändning av meddelande från styrenheten till CAN-nätverket, utan större framgång. Ytterst lite tid kunde ägnas åt denna frågeställning inom ramen för detta arbete.
Studeringar av kod från examensarbete [27] gjordes för att få en uppfattning om hur sändning och mottagning av meddelande går till. I nuläget var det endast sändning av meddelande som var intressant, men för projektet som helhet var det även bra att förstå hur mottagning av meddelande sker. Jämförelser med datablad [2] gjordes för att förstå logiken och samspelet mellan styrenhet och PICAN, men ingen egen kod gjordes för att testa sändning av meddelande. Däremot testades sändning av meddelande som beskrivs i avsnitt 5.4.5.

7.2 Krav

Slutprodukten är resultatet av de krav och önskemål som togs fram i början av projektet. Slutprodukten är ett fullt fungerande program anpassad för touchfunktionalitet. Ett användargränsit har tagits fram för att främja såväl HMI som ergonomi. Slutprodukten har lämnat utrymme åt vidareutveckling med flera menyalternativ och med utrymme för tillägg av funktioner i klimat- och belysningspanelen. Det går dessutom att ansluta fler alternativ till GrovePi som fuktsensor, ljudsensor, högtalare mm.
Kravet på haptisk feedback har implementerats men tyvärr med relativt svag funktion. Vibrationsmotorn som ger haptisk feedback är inte i nuläget den önskade. Idén om att införa haptisk feedback infördes sent i projektets gång, vilket inte gav utrymme nog för flertalet tester och förbättringar. Däremot ges tydlig digital feedback vid varje knapptryck i programmet. Styrning av funktioner via CAN-bus är den enda punkten som inte uppfylldes alls.

7.3 Avgränsningar

I tidig fas bestämdes klimat och belysning som huvudfokus och har förblivit så genom projektets gång. Diskussioner om ytterligare funktioner gjordes, men som inte hann implementeras.

8

Slutsats

Kunskaper inom elektronik och datateknik har varit till stor hjälp under projektets gång. Det som varit väldigt roligt och intressant med projektet är att en egen produkt arbetats fram för kund, i detta fall Volvo Bussar. Att försöka förstå kundens önskan samtidigt som man vill komma på egna idéer och lösningar har varit lärorikt i flera avseenden. Projektet har till största del innefattat Pythonprogrammering och felsökningar i både kod och mjukvara, men också den tekniska förståelsen av att sammanknyta ett helt projekt.

8.1 Diskussion av resultat

Vi anser att projektets arbete kan summeras som ett lyckat resultat med en slutprodukt som vi är nöjda med. Projektet har varit kostnadseffektivt med endast GrovePi som har varit en direkt kostnad.
Det enda kravet som inte uppfylldes var kommunikationen mellan programmet och CAN-bus. Dessvärre fanns det inte resurser från elsidan på Volvo Bussar i nuläget, trots upprepade uppmaningar från vår handledare. Deras medverkan hade behövts för att anpassa prototypen till en buss, som var vår vision. I och med att vi inte visste om de skulle medverka prioriterades istället annat.
Projektets avgränsningar har varierat under projektets gång, eftersom vi hade svårt att bedöma tidsåtgången. Ytterligare förslag på vidareutveckling:

- *Destination text.* Att kunna ställa in vad som ska stå på destinationsskylten.
- *Hållplats information.* Se vilka hållplatser som rutten innefattar, nästa hållplats och hur man ligger till i tiden.
- *Navigation för specifik rutt/linje.* En vägledning för chauffören att kunna se på navigatorn hur chauffören ska köra, för att komma till nästa hållplats.
- *Kameravy* över dörrarna vid på- och avstigning.
- *Backkamera.*
- *Linjekarta* för översikt av anslutande linjetrafik.
- *Väderrapport.*
- *Trafikmeddelande.* Få en trafikrapport i text och/eller tal.

8.2 Problem och begränsningar

Få mjukvarumoduler fungerade vid äldre version av operativsystemet. För att Raspberry-enheten skulle kunna samspela med mjukvarumoduler behövdes kontinuerlig uppdatering av mjukvaran. Uppdaterades inte mjukvaran kunde samspelet mellan kod och moduler bli fel, vilket kan leda till onödigt långa felsökningar.

Efter några veckor av arbete på Raspberry-enheten uppstod ett allvarligt fel, startpanelen i Raspbians gränssnitt började bete sig märkligt och strömförsörjning via USB-portarna slutade fungera. Efter flertalet felsökningar såg vi ingen annan utväg än att formatera SD-kortet med ett nytt operativsystem. Filerna kunde återställas från backups vi tagit, med endast några få förluster. Backup av all skriven kod sågs efter ominstallationen som en självklarhet till projektets slut.

Test av CAN-kommunikation simulerades i CANalyzer. Problemet med CANalyzer var att det var svårt att köra det på andra datorer än den stationära datorn som fanns på företaget. Installationen av CANalyzer gjordes på en laptop från Volvo, men gav felmeddelande direkt vid testsändning av meddelande. Som en följd av detta hade vi enbart tillgång till en dator på företaget med ett fungerande CANalyzerprogram.

Mjukvarumodulen Tkinter användes för all kod och de styrkor/svagheter som fastställts med mjukvarumodulen efter avslut av uppgift är:

- Hantering av bilder, etiketter och knappar i användargränssnittet sker på ett bra sätt där placering sker med x- och y-koordinater.
- Val av utseende på objekten i användargränssnittet är många.
- Bilder som importeras i användargränssnittet kan inte vara transparenta, problemet med detta är att etiketter och bilder ej kan placeras bakom varandra och synas.
- Tryckytan för knappar som skapas i användargränssnittet kan inte vara rund eller ha rundade kanter. Detta ger en utvecklingsbegränsning ifall mjuka kanter är en avgörande faktor.

8.3 Hållbar utveckling

Slutsatsen om vad projektet kan ha för nytta för ett hållbart samhälle återkopplas till vad prototypen kan ersätta. Mekaniska knappar och reglage till instrumentpanelerna minskas, vilket leder till minskad tillverkning av komponenter. Det kommer med hög sannolikhet gå åt mer energi att producera dessa komponenter för utökning av funktioner, med tanke på att instrumentpanelerna ser olika ut i många bussar. Tillökning av funktioner pågår ständigt som ställer krav på mer utrymme och nya komponenter. Med endast en touchdisplay som ersätter nuvarande komponenter/reglage blir energiåtgången mindre, eftersom samma produkt kan användas i alla bussar. Det som kan skilja

mellan touchdisplay-enheterna är dess utbytbara mjukvara, som beror på bussens syfte.
Rätt utformning av touchdisplayen kan öka säkerheten med enklare och mer koncentrerad instrumentpanel, som i sin tur kan leda till en bättre arbetsmiljö för föraren.

8.4 Saker som kunde gjorts annorlunda

Möten med Volvo Bussar gjordes först under projektets sjunde vecka, vi hade gärna sett att det skett tidigare. Synpunkter och önskemål hade kunnat åtgärdats snabbare och mer tid hade kunnat ägnats åt att jobba fram fler funktioner. Det handlade dock mer om brist på tid från Volvo Bussars sida än vår. Eloge till vår handledare som jobbade fram dessa möten.

Referenser

[1] Introduction to the Controller Area Network(CAN),

http://www.ti.com/lit/an/sloa101a/sloa101a.pdf (Acc 2016-10-28)

[2] MCP2515 Microship,

http://ww1.microchip.com/downloads/en/DeviceDoc/21801d.pdf(Acc 2016-10-07)

[3] Lundgren A (2016), *Visualization of Volvo Buses Chassis Electronic Systems via CAN-Bus*, Luleå: Luleå University of Technology (Acc 2016-11-17)

[4] CAN-H och CAN-L tillstånd,

http://www.analog.com/en/analog-dialogue/articles/circuit-for-adjustable-can-level-differential-output-signal.html(Acc 2016-11-17)

[5] Human Machine Interface (HMI),

http://www.anaheimautomation.com/manuals/forms/hmi-guide.php#sthash.v4LhWrNU.ajBl15CA.dpbs/(Acc 2016-11-06)

[6] ECE No.121, https://www.unece.org/fileadmin/DAM/trans/main/wp29/wp29regs/R121r1e.pdf(Acc 2016-11-20)

[7] ISO-16121.1:2012,

https://www.iso.org/obp/ui/#iso:std:iso:16121:-1:ed-2:v1:en (Acc 2016-10-11)

[8] Raspberry Pi,

https://www.raspberrypi.org/ (Acc 2016-10-11)

[9] Bild Raspberry Pi 2 model B,

https://www.pi-supply.com/wp-content/uploads/2015/02/100437_2_e7a28df2-0ef5-4423-94f3-7f04ebd9e6c1_1024x1024.jpg(Acc 2016-11-06)

[10] Raspbian,
https://www.raspberrypi.org/downloads/raspbian/ (Acc 2016-09-30)

[11] Bild PICAN,

http://skpang.co.uk/catalog/images/raspberrypi/pi_2/IMG_0002.jpg(Acc 2016-11-06)

[12] MCP2551 Microship,

http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf(Acc 2016-10-07)

[13] Bild GPIO,

https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/(Acc 2016-11-06)

[14] Bild DB9-kontakt,

http://infocenter.arm.com/help/topic/com.arm.doc.dui0163b/graphics/IM-PD1_serialBlock2.png(Acc 2016-11-06)

[15] DB9 till OBD2-kontakt,

http://img.tradera.net/medium/958/223817958_e2948923-9bfe-44c0-b38d-f7a4752dd5fb.jpg (Acc 2016-12-10)

[16] LCD-skärm,

https://www.chalk-elec.com/?page_id=1283#!/10-universal-LCD-with-HDMI-interface-and-capacitive-multi-touch/p/42545413/category=3094861(Acc 2016-10-28)

[17] GrovePi,

http://www.dexterindustries.com/shop/grovepi-board/(Acc 2016-10-28)

[18] Bild GrovePi,

http://img.tomtop-cdn.com/product/original/p/tt/e/0/e0333-7-3b5d.jpg(Acc 2016-11-06)

[19] GrovePi vibrationsmotor,

http://wiki.seeed.cc/Grove-Vibration_Motor/(Acc 2016-10-28)

[20] Bild Vibrationsmotor,

http://www.dexterindustries.com/wp-content/uploads/2016/04/gvibresized.png(Acc 2016-11-06)

[21] Python,

https://www.python.org/(Acc 2016-11-06)

[22] Interpreterande programspråk,

https://sv.wikipedia.org/wiki/Interpreterande_programspråk(Acc 2016-11-06)

[23] An Introduktion To Tkinter,

http://effbot.org/tkinterbook/tkinter-index.htm(Acc 2016-11-10)

[24] SPI - Serial Peripheral Interface,

http://www.corelis.com/education/SPI_Tutorial.htm/(Acc 2016-11-06)

[25] Bild SPI-kommunikation,

https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/SPI_single_slave.svg/350px-SPI_single_slave.svg.png(Acc 2016-11-06)

[26] Hansson C & Lagergren L (2016), *Omdesign av klimatpanel i Volvos bussar*, Göteborg: Chalmers tekniska högskola (Acc 2016-10-11)

[27] Påsse E & Holmstedt M (2015), *Prototyp för duplicering av instrumentpanel anpassad för övningskörning av buss*, Göteborg: Chalmers tekniska högskola (Acc 2016-11-11)

Bilaga 1

Programkod

Nedan prestenteras intressanta utdrag ur den programkod som använts: Upprepande kodrader med liknande funktion har tagits bort.

```
#!/usr/bin/python
# Import the Tkinter library to gain access to GUI components
from Tkinter import *
import Tkinter as tk
# Import the tkFont library to be able to create a custom font size
from tkFont import *
# Import from the PIL library to be able to use images on the buttons
from PIL import ImageTk, Image
import time
import grovepi

# Configurations for vibration motor
Vibration_motor = 8
grovepi.pinMode(Vibration_motor,"OUTPUT")

# Define some fonts
LARGE_FONT= ("Volvo Instrument 1", 12)
SMALL_FONT=("Volvo Instrument 1", 12)
SEAT_FONT= ("Volvo Instrument 1", 19)
MENU_FONT= ("Volvo Instrument 1", 22)
SUMLABEL_FONT= ("Volvo Instrument 1", 54)

class Displayscreen(tk.Tk):
###----------------------------Frame initial----------------------------###
    def __init__(self):

        tk.Tk.__init__(self)

        container = tk.Frame(self)
        container.pack(fill="both", side="top", expand = TRUE)  #Creating  frame
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}
###-------------------------Frame initial end-------------------------###
###-----------------------------Pages initial-----------------------------###
        for F in (StartPage, PageOne, PageTwo, PageThree, PageFour): #For more
 ↪  menus, add it into this

            frame = F(container, self)
            self.frames[F] = frame          #For each frame, set same proportion
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)   #Shows StartPage when program is started

    def show_frame(self, cont):    #Page handling function
```

```
46
47          frame = self.frames[cont]       #Selected page gets a created frame
    ↪   according to frame initalize
48          frame.tkraise()         #Places the selected page raised to top view
49  ###-------------------------Pages initial end-----------------------###
50  class PageOne(tk.Frame):
51
52      # Initiate function
53      def __init__(self, parent, controller):
54          tk.Frame.__init__(self,parent, background="black")
55  ###-------------------------------Fonts-------------------------------###
56          self.CelciusFont = Font("Volvo Instrument 1", size=30)
57          self.CelciusCircleFont = Font("Volvo Instrument 1", size =8)
58          self.PassengerDriverFont = Font("Volvo Instrument 1", size=18)
59          self.SumLabelFont = Font("Helvetica", size=55,)
60          self.PassengerDriverFont = Font("Helvetica", size=18)
61  ###-----------------------------Fonts end-----------------------------###
62
63  ###---------------------------Menubuttons---------------------------###
64    #Buttons to switch between pages
65
66        # Current page
67          button1 = tk.Button(self, text="Climate", command=lambda:
    ↪   controller.show_frame(StartPage), bg="black",
    ↪   relief="sunken",font=MENU_FONT, width=10, state=DISABLED, bd=5, height=2)
68          button1.place(relx=0.01, rely=0.030)
69
70          button2 = tk.Button(self, text="Interior light",
71                              command=lambda: controller.show_frame(PageOne),
    ↪   relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
72          button2.place(relx=0.17, rely=0.030)
73
74          button3 = tk.Button(self, text="Navigation",
75                              command=lambda: controller.show_frame(PageTwo),
    ↪   relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
76          button3.place(relx=0.33, rely=0.030)
77
78          button4 = tk.Button(self, text="Specifications",
79                              command=lambda: controller.show_frame(PageThree),
    ↪   relief="groove",font=MENU_FONT, width=11, bd=4, height=2)
80          button4.place(relx=0.49, rely=0.030)
81  ###-------------------------Menubuttons end-----------------------###
82
83  ###---------------------------Variables---------------------------###
84    # IntVar: Variable as integer
85    # DoubleVar: Same as IntVar, but with decimal
86    # StringVar: Variable that updates texts
87    # A DoubleVar is needed so that the GUI updates whenever the variable changes
88
89          self.TempPassengers = DoubleVar()
90          self.TempPassengers.set(20.0)    # Initial value
91
92          self.FanLevel = IntVar()
93          self.FanLevel.set(0)
94
95          self.MaxFanLevel = 5
```

```
96          self.MinFanLevel = 0
97
98          self.FanPosTop = StringVar()
99          self.FanPosTop.set("NEUTRAL")
100 ###----------------------------Variables end-----------------------###
101
102 ###-----------------------------Images-----------------------------###
103         self.AddFanImage = Image.open("Add.png")
104         self.AddFanImage = self.AddFanImage.resize((120, 120), Image.ANTIALIAS)
105         self.AddFanImage = ImageTk.PhotoImage(self.AddFanImage)
106 ###----------------------------Images end-------------------------###
107
108 ###----------------------------Buttons----------------------------###
109         # Command:  Run a function when pressed
110         # Relief:  3D button-style
111         # relx:  X-coordinates
112         # rely:  Y-coordinates
113         # anchor:  set a position (right, left, center etc)
114         # state: ON/OFF setting (NORMAL/DISABLED)
115
116         self.AddFanButton = Button(self, image=self.AddFanImage, bg="black",
    ↪  highlightbackground='black',activebackground="black", relief=FLAT)
117         self.AddFanButton.place(relx=.52, rely=0.82, anchor="c")
118         self.AddFanButton.config(width=120, height=120)
119         self.AddFanButton.bind('<ButtonPress-1>',self.AddFanButtonPress)
120         self.AddFanButton.bind('<ButtonRelease-1>',self.AddFanButtonRelease)
121
122         self.SubFanButton = Button(self, image=self.SubFanImage, bg="black",
    ↪  highlightbackground='black',activebackground="black",relief=FLAT)
123         self.SubFanButton.place(relx=.34, rely=0.82, anchor="c")
124         self.SubFanButton.config(width=120, height=120)
125         self.SubFanButton.bind('<ButtonPress-1>',self.SubFanButtonPress)
126         self.SubFanButton.bind('<ButtonRelease-1>',self.SubFanButtonRelease)
127 ###----------------------------Buttons end------------------------###
128
129 ###----------------------------Labels-----------------------------###
130         self.Fanbar1Label = Label(self, relief="ridge", bg="black", width=3,
    ↪  height=5, bd=5) #Fanbar1
131         self.Fanbar1Label.place(relx=.330, rely=0.72, anchor="s")
132
133         self.Fanbar2Label = Label(self, relief="ridge", bg="black", width=3,
    ↪  height=10, bd=5) #Fanbar2
134         self.Fanbar2Label.place(relx=.380, rely=0.72, anchor="s")
135
136         self.Fanbar3Label = Label(self, relief="ridge", bg="black", width=3,
    ↪  height=15, bd=5) #Fanbar3
137         self.Fanbar3Label.place(relx=.430, rely=0.72, anchor="s")
138
139         self.Fanbar4Label = Label(self, relief="ridge", bg="black", width=3,
    ↪  height=20, bd=5) #Fanbar4
140         self.Fanbar4Label.place(relx=.480, rely=0.72, anchor="s")
141
142         self.Fanbar5Label = Label(self, relief="ridge", bg="black", width=3,
    ↪  height=25, bd=5) #Fanbar5
143         self.Fanbar5Label.place(relx=.530, rely=0.72, anchor="s")
144
```

```
145         self.PassengerLabel = Label(self, text="Passengers", relief=FLAT,
    ↪  bg="black", fg="white", font=SEAT_FONT)
146         self.PassengerLabel.place(relx=.91, rely=.17, anchor="s")
147 
148         self.DriverLabel = Label(self, text="Driver", relief=FLAT, bg="black",
    ↪  fg="white", font=SEAT_FONT)
149         self.DriverLabel.place(relx=.70, rely=.17, anchor="s")
150 
151         self.CelciusLabelRight = Label(self, text ="o", fg="white", bg="black",
    ↪  font= self.CelciusCircleFont, relief=FLAT)
152         self.CelciusLabelRight.place(relx=0.96, rely=0.15)
153 
154         self.CelciusLabelLeft = Label(self, text ="o", fg="white", bg="black",
    ↪  font= self.CelciusCircleFont, relief=FLAT)
155         self.CelciusLabelLeft.place(relx=0.75, rely=0.15)
156 ###----------------------------Labels end----------------------------###
157 
158 ###-----------------------------Scales-----------------------------###
159         self.TempScalePassengers = Scale(self, showvalue=0, bg="grey",
    ↪  activebackground="grey", sliderlength=55, command=self.TempSliderPassengers,
    ↪  resolution= 0.15, highlightbackground='white', highlightcolor="black",
    ↪  from_=18, to=25, length=500, troughcolor="black", relief=FLAT,
    ↪  sliderrelief=SOLID, width = 120)
160         self.TempScalePassengers.place(relx=.92, rely=.95, anchor= "s")
161         self.TempScalePassengers.set(18.0)
162         self.TempScalePassengers.bind('<B1-Motion>',
    ↪  self.TempScalePassengersButtonMotion)
163         self.TempScalePassengers.bind('<ButtonPress-1>',
    ↪  self.TempScalePassengersButtonPress)
164         self.TempScalePassengers.bind('<ButtonRelease-1>',
    ↪  self.TempScalePassengersButtonRelease)
165 ###-------------------------Scales end----------------------------###
166 
167 ###--------------------------Functions----------------------------###
168  # self."variable".set():   Change a variables value
169  # self."variable".config():  Chage parameters for labels/buttons
170  # self."variable".get():   Current value of the variable
171  # "program".after("time in ms","function"): Runs a function after a specified
    ↪   time
172  # "program".after_cancel:   Stops a command from happen
173 
174     # Makes vibration for 0.8 sec whenever a button is pushed
175     def Vibration(self):
176         grovepi.digitalWrite(Vibration_motor,1)
177         time.sleep(0.08)
178         grovepi.digitalWrite(Vibration_motor,0)
179         pass
180 ###---------------------Temperature scales---------------------###
181     # Enable scrolling the slider up and down
182     def TempScalePassengersButtonMotion(self, event):
183       self.TempScalePassengers.config(state=NORMAL)
184       app.after_cancel(self.ButtonHolded.get())
185 
186     # Sets the specified temperature if someone clicks between 18-25
187     def TempScalePassengersButtonPress(self, event):
188       self.ButtonHolded.set(app.after(100, self.SetScalePassengers))
```

```
189
190     # Defines between which coordinates the user clicked on the scale, and sets
    ↪  the temperature.
191     def SetScalePassengers(self):
192       self.Vibration()
193       # Minimum distance from top=227
194       # Maximum distance from top=728
195       y_coord = app.winfo_pointery() - app.winfo_rooty() #Mouse x-coordinates
    ↪  depending on the size of the screen
196       if y_coord >= 227 and y_coord < 292:
197        self.TempScalePassengers.set(18.0)
198       elif y_coord >=292 and y_coord < 354:
199        self.TempScalePassengers.set(19.0)
200       elif y_coord >=354 and y_coord < 416:
201        self.TempScalePassengers.set(20.0)
202       elif y_coord >=416 and y_coord < 478:
203        self.TempScalePassengers.set(21.0)
204       elif y_coord >=478 and y_coord < 540:
205        self.TempScalePassengers.set(22.0)
206       elif y_coord >=540 and y_coord < 602:
207        self.TempScalePassengers.set(23.0)
208       elif y_coord >=602 and y_coord < 664:
209        self.TempScalePassengers.set(24.0)
210       elif y_coord >=664 and y_coord < 728:
211        self.TempScalePassengers.set(25.0)
212
213       self.TempScalePassengers.config(state=DISABLED) # Stops the slider from
    ↪  increasing/decreasing constantly
214
215     # Sets state to NORMAL when finger is realeased
216     def TempScalePassengersButtonRelease(self, event):
217       self.TempScalePassengers.config(state=NORMAL)
218
219     # Updates sumlabel for temperature for the right scale
220     def TempSliderPassengers(self, ValueFromSlider):
221       # Round of the every 0.5 value
222       TempValueBeforeRev = round(self.TempScalePassengers.get() * 2) / 2
223
224       # Reverse order of temperature of the slider
225       if TempValueBeforeRev == 18:
226        self.TempPassengers.set(25.0)
227       elif TempValueBeforeRev == 18.5:
228        self.TempPassengers.set(24.5)
229       elif TempValueBeforeRev == 19.0:
230        self.TempPassengers.set(24.0)
231       elif TempValueBeforeRev == 19.5:
232        self.TempPassengers.set(23.5)
233       elif TempValueBeforeRev == 20.0:
234        self.TempPassengers.set(23.0)
235       elif TempValueBeforeRev == 20.5:
236        self.TempPassengers.set(22.5)
237       elif TempValueBeforeRev == 21.0:
238        self.TempPassengers.set(22.0)
239       elif TempValueBeforeRev == 21.5:
240        self.TempPassengers.set(21.5)
241       elif TempValueBeforeRev == 22.0:
```

```
242         self.TempPassengers.set(21.0)
243       elif TempValueBeforeRev == 22.5:
244         self.TempPassengers.set(20.5)
245       elif TempValueBeforeRev == 23.0:
246         self.TempPassengers.set(20.0)
247       elif TempValueBeforeRev == 23.5:
248         self.TempPassengers.set(19.5)
249       elif TempValueBeforeRev == 24.0:
250         self.TempPassengers.set(19.0)
251       elif TempValueBeforeRev == 24.5:
252         self.TempPassengers.set(18.5)
253       elif TempValueBeforeRev == 25.0:
254         self.TempPassengers.set(18.0)
255 ###----------------------Temperature scales end--------------------###
256
257 ###---------------------------Fan levels---------------------------###
258     # Maximize fanlevel if button is pressed more then 400ms
259     def AddFanButtonPress(self,event):
260       self.ButtonHolded.set(app.after(400, self.SetFanToMax))
261
262     # Maximize fanlevel
263     def SetFanToMax(self):
264       self.Vibration()
265       self.FanLevel.set(self.MaxFanLevel)
266       self.Add_Fan()
267
268     # Cancel maximum level if button is realeased within 400ms, and increase
    ↪  fanlevel if its not already maximized
269     def AddFanButtonRelease(self, event):
270       app.after_cancel(self.ButtonHolded.get())
271       if self.FanLevel.get() >= self.MaxFanLevel:
272    pass
273   else:
274    self.Add_Fan()
275
276       # Increases fanlevel, or passing if its already maximized
277     def Add_Fan(self):
278       self.Vibration()
279       if self.FanLevel.get() >= self.MaxFanLevel:
280         self.Fanbar1Label.config(bg="lightgrey")
281         self.Fanbar2Label.config(bg="lightgrey")
282         self.Fanbar3Label.config(bg="lightgrey")
283         self.Fanbar4Label.config(bg="lightgrey")
284         self.Fanbar5Label.config(bg="lightgrey")
285         pass
286       else:
287         self.FanLevel.set(self.FanLevel.get()+1)
288         if self.FanLevel.get() == 1:
289           self.Fanbar1Label.config(bg="lightgrey")
290         elif self.FanLevel.get() == 2:
291           self.Fanbar2Label.config(bg="lightgrey")
292         elif self.FanLevel.get() == 3:
293           self.Fanbar3Label.config(bg="lightgrey")
294         elif self.FanLevel.get() == 4:
295           self.Fanbar4Label.config(bg="lightgrey")
296         elif self.FanLevel.get() == 5:
```

```
297         self.Fanbar5Label.config(bg="lightgrey")
298 
299     # Minimize fanlevel if button is pressed more then 400ms
300     def SubFanButtonPress(self,event):
301       self.ButtonHolded.set(app.after(400, self.SetFanToMin))
302 
303     # Minimize fanlevel
304     def SetFanToMin(self):
305         self.Vibration()
306         self.FanLevel.set(self.MinFanLevel)
307         self.Sub_Fan()
308 
309     # Cancel minimum level if button is realeased within 400ms, and decrease
    ↪  fanlevel if its not already minimized
310     def SubFanButtonRelease(self, event):
311       app.after_cancel(self.ButtonHolded.get())
312       if self.FanLevel.get() <= self.MinFanLevel:
313        pass
314       else:
315        self.Sub_Fan()
316 
317     # Decreases fanlevel, or passing if its already minimized
318     def Sub_Fan(self):
319       self.Vibration()
320       if self.FanLevel.get() <= self.MinFanLevel:
321        self.Fanbar1Label.config(bg="black")
322        self.Fanbar2Label.config(bg="black")
323        self.Fanbar3Label.config(bg="black")
324        self.Fanbar4Label.config(bg="black")
325        self.Fanbar5Label.config(bg="black")
326        pass
327       else:
328        self.FanLevel.set(self.FanLevel.get()-1)
329        if self.FanLevel.get() == self.MinFanLevel:
330         self.Fanbar1Label.config(bg="black")
331        elif self.FanLevel.get() == 1:
332         self.Fanbar2Label.config(bg="black")
333        elif self.FanLevel.get() == 2:
334         self.Fanbar3Label.config(bg="black")
335        elif self.FanLevel.get() == 3:
336         self.Fanbar4Label.config(bg="black")
337        elif self.FanLevel.get() == 4:
338         self.Fanbar5Label.config(bg="black")
339 ###-------------------------Fan levels end-------------------------###
340 
341 ###-----------------------------Functions end-------------------------###
342 class PageTwo(tk.Frame):
343 
344  def __init__(self, parent, controller):
345   tk.Frame.__init__(self, parent, background="black")
346 ###-------------------------------Fonts-------------------------------###
347   self.SwitchInformationFont = Font("Volvo Instrument 1", size=12)
348   self.SwitchInformationFont = Font("Volvo Instrument 1", size=14)
349 ###-----------------------------Fonts end-----------------------------###
350 
351 ###-----------------------------Menubuttons---------------------------###
```

```
352   button1 = tk.Button(self, text="Climate",
353         command=lambda: controller.show_frame(StartPage),
      ↪  relief="groove",font=MENU_FONT,width=10, bd=4, height=2)
354   button1.place(relx=0.01, rely=0.030)
355
356   button2 = tk.Button(self, text="Interior light", # Current page
357         command=lambda: controller.show_frame(PageOne),
      ↪  bg="black",font=MENU_FONT,width=10, relief="sunken", state=DISABLED, bd=5,
      ↪  height=2)
358   button2.place(relx=0.17, rely=0.030)
359
360   button3 = tk.Button(self, text="Navigation",
361         command=lambda: controller.show_frame(PageTwo),
      ↪  relief="groove",font=MENU_FONT,width=10, bd=4, height=2)
362   button3.place(relx=0.33, rely=0.030)
363
364   button4 = tk.Button(self, text="Specifications",
365         command=lambda: controller.show_frame(PageThree),
      ↪  relief="groove",font=MENU_FONT,width=11, bd=4, height=2)
366   button4.place(relx=0.49, rely=0.030)
367 ###----------------------------Menubuttons end-------------------------###
368
369 ###-----------------------------Variables------------------------------###
370   # IntVar: Variable as integer
371   # DoubleVar: Same as IntVar, but with decimal
372   # StringVar: Variable that updates texts
373   # A DoubleVarVar is needed so that the GUI updates whenever the variable
      ↪  changes
374
375   self.LightPos = StringVar()
376   self.LightPos.set("NEUTRAL")
377 ###----------------------------Variables end-------------------------###
378
379 ###-------------------------------Images-------------------------------###
380   self.DimeLightImage = Image.open("dime_light.png")
381   self.DimeLightImage = self.DimeLightImage.resize((130, 130), Image.ANTIALIAS)
382   self.DimeLightImage = ImageTk.PhotoImage(self.DimeLightImage)
383
384   self.FullLightImage = Image.open("Interior_full_light.png")
385   self.FullLightImage = self.FullLightImage.resize((130, 130), Image.ANTIALIAS)
386   self.FullLightImage = ImageTk.PhotoImage(self.FullLightImage)
387 ###----------------------------Images end----------------------------###
388
389 ###-----------------------------Buttons--------------------------------###
390   self.DimeLightButton = Button(self, image=self.DimeLightImage,
      ↪  bg="black",activebackground="black", highlightbackground='black',
      ↪  command=self.DimeLight, relief=FLAT)
391   self.DimeLightButton.place(relx=0.41, rely=0.468, anchor="c")
392   self.DimeLightButton.config(width=128, height=125)
393
394   self.FullLightButton = Button(self, image=self.FullLightImage,
      ↪  bg="black",activebackground="black",
      ↪  highlightbackground='black',command=self.FullLight, relief=FLAT)
395   self.FullLightButton.place(relx=.41, rely=0.298, anchor="c")
396   self.FullLightButton.config(width=128, height=125)
397
```

```
398     self.SwitchInformation = Button(self, text="Switch information",
        ↪  bg="lightgrey", relief="solid", command=self.SwitchInformationPress,
        ↪  width=23, height=2, font=self.SwitchInformationFont, bd=2)
399     self.SwitchInformation.place(relx=.11, rely=0.9, anchor="c")
400   ###-----------------------------Buttons end---------------------------###
401
402   ###-------------------------------Labels------------------------------###
403     self.FullLightLabel = Label(self, text="Full Light", fg="black", bg="black",
        ↪  font=self.SwitchInformationFont, relief=FLAT)
404     self.FullLightLabel.place(relx=.41, rely=.2, anchor="s")
405
406     self.DimeLightLabel = Label(self, text="Dimmed Light", fg="black", bg="black",
        ↪  font=self.SwitchInformationFont, relief=FLAT)
407     self.DimeLightLabel.place(relx=.41, rely=.6, anchor="s")
408   ###-----------------------------Labels end----------------------------###
409
410   ###----------------------------Functions----------------------------###
411
412    # Makes vibration for 0.8 sec whenever a button is pushed
413    def Vibration(self):
414     grovepi.digitalWrite(Vibration_motor,1)
415     time.sleep(0.08)
416     grovepi.digitalWrite(Vibration_motor,0)
417     pass
418
419    # Change foreground color of each switchlabel to white when pressed. Active in 5
       ↪  seconds
420    def SwitchInformationPress(self):
421     self.FullLightLabel.config(fg="white")
422     self.DimeLightLabel.config(fg="white")
423     self.FullDriversLightLabel.config(fg="white")
424     self.AutoDriversLightLabel.config(fg="white")
425     self.FullInteriorLightFrontLabel.config(fg="white")
426     self.AutoInteriorLightFrontLabel.config(fg="white")
427     self.IndividualReadingLightLabel.config(fg="white")
428     self.AllReadingLightLabel.config(fg="white")
429     self.AutoInteriorLightRearLabel.config(fg="white")
430     self.FullInteriorLightRearLabel.config(fg="white")
431     self.DLChoiceLabel.config(fg="white")
432     self.NightLightLabel.config(fg="white")
433     self.FloorLightningLabel.config(fg="white")
434     self.RightTubeLabel.config(fg="white")
435    # Timer function for switchinformation released
436     app.after(5000, self.SwitchInformationOff)
437
438    # Change foregroundcolor of each switchlabel to black
439    def SwitchInformationOff(self):
440     self.FullLightLabel.config(fg="black")
441     self.DimeLightLabel.config(fg="black")
442     self.FullDriversLightLabel.config(fg="black")
443     self.AutoDriversLightLabel.config(fg="black")
444     self.FullInteriorLightFrontLabel.config(fg="black")
445     self.AutoInteriorLightFrontLabel.config(fg="black")
446     self.IndividualReadingLightLabel.config(fg="black")
447     self.AllReadingLightLabel.config(fg="black")
448     self.AutoInteriorLightRearLabel.config(fg="black")
```

```
449     self.FullInteriorLightRearLabel.config(fg="black")
450     self.DLChoiceLabel.config(fg="black")
451     self.NightLightLabel.config(fg="black")
452     self.FloorLightningLabel.config(fg="black")
453     self.RightTubeLabel.config(fg="black")
454 
455     # Topswitch full light
456     def FullLight(self):
457         self.Vibration()
458         if self.LightPos.get() != "BOT":
459             self.FullLightButton.config(bg="white", activebackground="white", bd=7)
460             self.DimeLightButton.config(bg="black", activebackground="black", relief=FLAT,
    ↪  bd=0)
461             self.LightPos.set("BOT")
462         elif self.LightPos.get() == "BOT":
463             self.FullLightButton.config(bg="black", activebackground="black", bd=0)
464             self.LightPos.set("NEUTRAL")
465 
466     # Botswitch dimmed light
467     def DimeLight(self):
468         self.Vibration()
469         if self.LightPos.get() != "TOP":
470             self.DimeLightButton.config(bg="white", activebackground="white", bd=7)
471             self.FullLightButton.config(bg="black", activebackground="black", relief=FLAT,
    ↪  bd=0)
472             self.LightPos.set("TOP")
473         elif self.LightPos.get() == "TOP":
474             self.DimeLightButton.config(bg="black", activebackground="black", bd=0)
475             self.LightPos.set("NEUTRAL")
476 ###-----------------------------Functions end-------------------------###
477 class PageThree(tk.Frame):
478 
479     def __init__(self, parent, controller):
480         tk.Frame.__init__(self, parent, background="black")
481 ###-------------------------------Menubuttons-----------------------------###
482         button1 = tk.Button(self, text="Climate",
483                             command=lambda: controller.show_frame(StartPage),
    ↪  relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
484         button1.place(relx=0.01, rely=0.030)
485 
486         button2 = tk.Button(self, text="Interior light",
487                             command=lambda: controller.show_frame(PageOne),
    ↪  relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
488         button2.place(relx=0.17, rely=0.030)
489 
490         button3 = tk.Button(self, text="Navigation",
491                             command=lambda: controller.show_frame(PageTwo),
    ↪  bg="black",font=MENU_FONT, width=10, relief="sunken", state=DISABLED, bd=5,
    ↪  height=2)
492         button3.place(relx=0.33, rely=0.030)
493 
494         button4 = tk.Button(self, text="Specifications",
495                             command=lambda: controller.show_frame(PageThree),
    ↪  relief="groove",font=MENU_FONT, width=11, bd=4, height=2)
496         button4.place(relx=0.49, rely=0.030)
497 ###---------------------------Menubuttons end---------------------------###
```

```
498
499 ###---------------------------------Images---------------------------------###
500         self.NavigationImage = Image.open("navigation.png")
501         self.NavigationImage = self.NavigationImage.resize((1050, 600),
    ↪  Image.ANTIALIAS)
502         self.NavigationImage = ImageTk.PhotoImage(self.NavigationImage)
503 ###-------------------------------Images end-------------------------------###
504 ###---------------------------------Labels---------------------------------###
505         self.Navigation = Label(self, image=self.NavigationImage,
    ↪  relief="raised", bg="black", bd=4)
506         self.Navigation.place(relx=0.5, rely=0.988, anchor="s")
507 ###-------------------------------Labels end-------------------------------###
508 class PageFour(tk.Frame):
509
510  def __init__(self, parent, controller):
511   tk.Frame.__init__(self, parent, background="black")
512 ###-----------------------------Menubuttons------------------------------###
513   button1 = tk.Button(self, text="Climate",
514                       command=lambda: controller.show_frame(StartPage),
    ↪  relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
515   button1.place(relx=0.01, rely=0.030)
516
517   button2 = tk.Button(self, text="Interior light",
518                       command=lambda: controller.show_frame(PageOne),
    ↪  relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
519   button2.place(relx=0.17, rely=0.030)
520
521   button3 = tk.Button(self, text="Navigation",
522                       command=lambda: controller.show_frame(PageTwo),
    ↪  relief="groove",font=MENU_FONT, width=10, bd=4, height=2)
523   button3.place(relx=0.33, rely=0.030)
524
525   button4 = tk.Button(self, text="Specifications",
526       command=lambda: controller.show_frame(PageThree),
    ↪  bg="black",font=MENU_FONT, width=11, relief="sunken", state=DISABLED, bd=5,
    ↪  height=2)
527   button4.place(relx=0.49, rely=0.030)
528
529   button5 = tk.Button(self, text="Included functions",
530                       command=lambda: controller.show_frame(PageFour),
    ↪  relief="groove", bd=3,font=LARGE_FONT, width=20, height=2)
531   button5.place(relx=0.03, rely=0.91)
532 ###-----------------------------Menubuttons end--------------------------###
533
534 ###---------------------------------Images---------------------------------###
535   self.BusProfilImage = Image.open("BusProfil.png")
536   self.BusProfilImage = self.BusProfilImage.resize((520, 250), Image.ANTIALIAS)
537   self.BusProfilImage = ImageTk.PhotoImage(self.BusProfilImage)
538
539   self.BusMeasureImage = Image.open("BusMeasure.png")
540   self.BusMeasureImage = self.BusMeasureImage.resize((520, 250), Image.ANTIALIAS)
541   self.BusMeasureImage = ImageTk.PhotoImage(self.BusMeasureImage)
542
543   self.BusWeightImage = Image.open("BusWeight.png")
544   self.BusWeightImage = self.BusWeightImage.resize((520, 85), Image.ANTIALIAS)
545   self.BusWeightImage = ImageTk.PhotoImage(self.BusWeightImage)
```

```
546
547     self.BusEngineImage = Image.open("BusEngine.png")
548     self.BusEngineImage = self.BusEngineImage.resize((520, 70), Image.ANTIALIAS)
549     self.BusEngineImage = ImageTk.PhotoImage(self.BusEngineImage)
550
551     self.BusGearsImage = Image.open("BusGears.png")
552     self.BusGearsImage = self.BusGearsImage.resize((520, 85), Image.ANTIALIAS)
553     self.BusGearsImage = ImageTk.PhotoImage(self.BusGearsImage)
554
555     self.BusSteeringImage = Image.open("BusSteering.png")
556     self.BusSteeringImage = self.BusSteeringImage.resize((520, 75),
    ↪  Image.ANTIALIAS)
557     self.BusSteeringImage = ImageTk.PhotoImage(self.BusSteeringImage)
558   ###------------------------------Images end------------------------------###
559
560   ###------------------------------Labels------------------------------###
561     self.BusProfil = Label(self, image=self.BusProfilImage, relief="raised",
    ↪  bg="black", bd=4)
562     self.BusProfil.place(relx=0.75, rely=0.55, anchor="s")
563
564     self.BusMeasure = Label(self, image=self.BusMeasureImage, relief="raised",
    ↪  bg="black", bd=4)
565     self.BusMeasure.place(relx=0.25, rely=0.55, anchor="s")
566
567     self.BusWeight = Label(self, image=self.BusWeightImage, relief="raised",
    ↪  bg="black", bd=4)
568     self.BusWeight.place(relx=0.75, rely=0.7, anchor="s")
569
570     self.BusEngine = Label(self, image=self.BusEngineImage, relief="raised",
    ↪  bg="black", bd=4)
571     self.BusEngine.place(relx=0.75, rely=0.85, anchor="s")
572
573     self.BusGears = Label(self, image=self.BusGearsImage, relief="raised",
    ↪  bg="black", bd=4)
574     self.BusGears.place(relx=0.25, rely=0.7, anchor="s")
575
576     self.BusSteering = Label(self, image=self.BusSteeringImage, relief="raised",
    ↪  bg="black", bd=4)
577     self.BusSteering.place(relx=0.25, rely=0.85, anchor="s")
578   ###------------------------------Labels end------------------------------###
579   class PageFive(tk.Frame):
580    def __init__(self, parent, controller):
581     tk.Frame.__init__(self, parent, background="black")
582   ###------------------------------Menubuttons------------------------------###
583     button1 = tk.Button(self, text="Back", command=lambda:
    ↪  controller.show_frame(PageThree),relief="groove",bd=3,font=LARGE_FONT,
    ↪  width=20, height=2)
584     button1.place(relx=0.03, rely=0.91)
585   ###------------------------------Menubuttons end------------------------------###
586
587   ###------------------------------Images------------------------------###
588     self.FunctionImage = Image.open("funktioner.jpg")
589     self.FunctionImage = self.FunctionImage.resize((650, 650), Image.ANTIALIAS)
590     self.FunctionImage = ImageTk.PhotoImage(self.FunctionImage)
591
```

```
592   self.Functions = Label(self, image=self.FunctionImage, relief="raised",
      ↪  bg="black", bd=4)
593   self.Functions.place(relx=0.5, rely=0.98, anchor="s")
594 ###------------------------------Images end------------------------------###
595
596 ###----------------------------------Main----------------------------------###
597 if __name__ == '__main__':
598
599  app = Displayscreen()
600  app.title("Displayscreen")          #Headername of window
601  app.geometry("1366x768")            #Size of window when program is started
602  app.minsize(1366,768)               #Maximum screensize (makes it unable to
     ↪  resize the screensize while program is executed)
603  app.maxsize(1366,768)               #Minimum screensize (makes it unable to
     ↪  resize the screensize while program is executed)
604  #app.config(cursor='none')          #Uncomment to hide mousepointer
605  app.attributes('-fullscreen', True) #Fullscreenmode
606  app.mainloop()                      #Everything past this will not be executed
607 ###----------------------------------Main end------------------------------###
```