CHALMERS
UNIVERSITY OF TECHNOLOGY

Multi- kamera interlink för birdview

Multiple-camera interlink for birdview

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

Mattias Bengtsson
Patrik Wallin

Institutionen för Signaler och System
Chalmers tekniska högskola
Göteborg, Sverige 2016

EXAMENSARBETE 2016

Multi- kamera interlink för birdview

Multiple-camera interlink for birdview

Mattias Bengtsson
Patrik Wallin

CHALMERS

Institutionen för signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Förord

Genom detta examensarbete har vi fått möjligheten att applicera kunskap inom områden som vi berört vid högskoleutbildningen på Chalmers tekniska universitet. Detta arbete på 15hp är skrivit för instutionen signaler och system.

Arbetet har utförts av två studenter på Mekatronikprogrammet 180hp åt företaget Benteler Engineering, ett konsultbolag som till stor del är riktat mot fordonsindustrin. Mestadels tid har lagts ner på Bentelers kontor på Nya varvet, västra Frölunda. Arbetet berörde olika ingenjörsdelar men framförallt områden elektro- och datavetenskap.

Vi vill tacka företaget Benteler Engineering och vår handledare David som gett oss chansen att visa vad vi kan.

Tack också till Manne Stenberg, vår handledare på Chalmers, Bertil Thomas vår examinator och även ett tack till Sakib Sistek för hjälp med 3D skrivning.

Göteborg 2016-06-02

Mattias Bengtsson & Patrik Wallin

Sammanfattning

Att ha koll på sin omgivning ifrån sitt fordon är en självklarhet för att manövrera sig utan problem. Ett hjälpmedel för att få bättre vy av vad som ligger runt bilen, vid parkering och snäva miljöer är önskat.

Projektet handlar om att montera fyra stycken vidvinkelkameror på en bil som täcker alla ytor runt fordonet och sen hantera denna videodata. Att hantera videodata i realtid med stor dataöverföring och väva ihop videovyerna till en enda realistisk bild är svårigheten i detta projekt. Den ihopvävda bilden ska alltså ge en vy som simulerar ”birdview”, bilen och omgivningen uppifrån.

Arbetet har utförts tillsammans med företaget Benteler Engineering och en enklare prototyp av grundidén har tagits fram. Resultatet är tänkt som ”proof of concept” och har budgetbegräsningar vilket direkt påverkar val av hårdvara. Detta har i sin tur lett till att utnyttja prestanda maximalt.

Abstract

To be aware of your surroundings as a driver is a given for maneuver your vehicle without a problem. A equipment to give the driver a better view of the cars surroundings, in case of parking and narrow environments is required.

The task in this project is to mount four wide-angle cameras on a car that covers all the surroundings around the vehicle and then handle this videodata appropriately. To handle videodata in realtime with that amount of datatransfer and to videostich the views to a realistic image is the main difficulty in this project. Accordingly the stitched image is going to show a simulated “birdview”, the vehicle and surroundings from above.

The project has been produced and completed alongside with Benteler engineering and a basic prototype from the original idea has been designed. The product itself is thought as a “proof of concept” and has both budget restrains that directly affect hardware choice. This has led to a path that as effectively as possible uses performance.

Innehåll

Beteckningar

RPi	–	Raspberry pi
C	–	Kodspråk
Fps	–	Frames per second
MP	–	Megapixel
I/O	–	Input/output
GPIO	–	General purpose input/output
OS	–	Operativsystem
IDE	–	Intergrated Development Enviroment
FFC	–	Flexible flat cable
FRC	–	Flexible round cable
OpenCV	–	Open source computer vision library
Catia V5	–	Cad program
Photoshop	–	Fotoediteringsprogram
Premier	–	Videoediteringsprogram
PMMA	–	Termoplasten polymetylmetalkrylat
Numpy	–	Numeriskt beräkningspaket till python
Virtuellenv	–	Virtuell arbetsmiljö för konfigurering
Frame	–	Intagen stillbild ifrån kameran
STL	–	STereoLithography

Figurer

1. Inledning

1.1 Bakgrund

Benteler Engineerings avsikt med ex-jobbet var tydlig från början då ett aktuellt projekt kopplat till deras arbetsområde kan i framtiden användas som argument för att få in nya konsulter på uppdrag. Vår ambition var att ha ett inspirerande och utmanande projekt och efter möten togs en idé fram som var av intresse för båda parter.

1.2 Syfte

Med hjälp av ingenjörsmässiga metoder ta fram ett ”proof of concept” för en aktuell produkt till bilindustrin.

1.3 Mål

Målet är att tillverka en videotillämpning till motorfordon, som för samman fyra stycken vidvinkelkameror i realtid och skapar en överblick kring fordonet i ”birdview” format.

1.4 Avgränsningar

Applicering av sensorer och avståndshantering begränsas bort. Projektet ska vara färdigt för att presenteras både muntligt och skriftligt inom projektets tidsplan. Projektets deadline ligger på 10 veckor och redovisning sker den 10 juni 2016. Ekonomiska resurser för hårdvara är begränsat.

1.5 Precisering av frågeställningen

- Klarar enklare hårdvara av att väva ihop flera kameror?
- Går det att visa en realistisk bild på hanterad video utan fördröjningar?
- Är det möjligt att implementera i verklig miljö?

1.6 Resursplan

1.6.1 Personer

Projektet utförs av studenterna Mattias Bengtsson och Patrik Wallin.
Handledare från Benteler är David Anberg och från Chalmers Manne Stenberg.
Examinator för projektet är Bertil Thomas från institutionen Signaler och System.

1.6.2 Material

Hårdvara ansvarar studenterna för att införskaffa, med godkännande av företagshandledare, Benteler ersätter detta.

1.6.3 Lokaler

Lindholmens lokaler kommer nyttjas vid hårdvarebygge men kodning och projektdokumentation kommer i största utsträckning ske vid Bentelers lokaler vid nya varvet.

2. Teknisk Bakgrund

2.1 Hårdvara

2.1.1 Raspberry Pi 3 Model B

Raspberry Pi är en enkortsdator i samma storlek som ett kreditkort. Idén kom 2006 för att öka intresset omkring datorer och programmering, som sedan slutet av 90-talet hade varit stadigt sjunkande. Tanken var att den skulle användas i både utbildningssyfte och som leksak i hemmet. Efter hårt arbete kunde The Raspberry Pi Foundation år 2012 släppa RPi 1. Under kommande år, med mycket hjälp från engagerade programmerare runt om i världen, har anställda utvecklat sin produkt och släppa versionen som kommer användas i detta projekt, Raspberry Pi 3 Model B[1].

Processor	1.2GHz 64-bit quad-core ARMv8 CPU
Grafikprocessor	VideoCore IV 3D graphics core
RAM	1 GB
Kameragränssnitt	CSI
Videogränssnitt	DSI
HDMI	Ja
USB	4 st 2.0
GPIO	40 st
Nätvärksuttag	Ja
Ljudutgång	3.5 mm
WiFi	Ja
Bluetooth	4.1
Bluetooth low energy	Ja
SD	Micro

Figur 1 Raspberry pi 3 model B specifikation

2.1.2 CSI

RPi har ett eget överföringsgränssnitt för just kameror, CSI (camera serial interface). Via FFC (flexible flat kabel) med 15 pinnar kopplas kamera direkt in på raspberry kortet. Då dessa kablar är relativt sköra görs en konvertering till FRC (flexible round cable). CSI gränssnittet togs fram för mobilindustrin då man ville ha snabbare och stabilare dataöverföring, vilket har lett till att högre upplösning kan användas. Storlek har också haft betydelse vid framtagandet och nästan papperstunna kablar används[9].

2.1.3 DSI

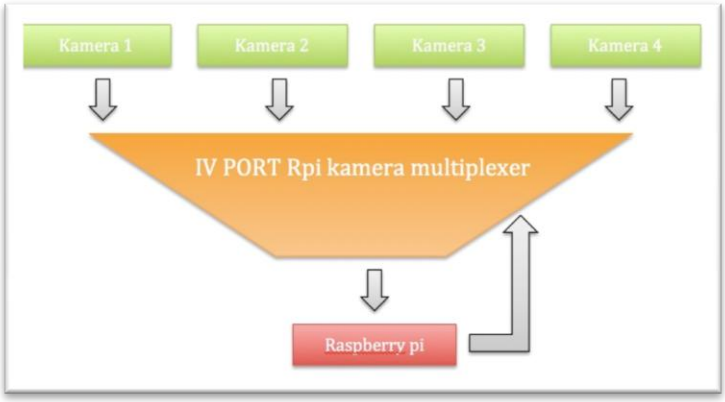
RPi:n använder sig först och främst av två stycken gränssnitt när det kommer till överföring av bild till skärm. Det ena är det välkända HDMI och det andra är det mindre kända DSI(Display Serial Interface). DSI är ett gränssnitt som drar väldigt lite ström och skapades, precis som CSI, för ett specifikt användningsområde. Detta gör att man inte hittar DSI på många andra ställen än i mobiltelefoner[6].

2.1.4 Kameror

Raspberry pi anpassade kameror med CSI anslutning har hög prestanda med tanke på priset. Kameraplattan, som endast är 2,5x2,5 cm, har en 5 MP kamera som kan hantera 1080p med 30 fps. Vidvinkel är 160° och huset har fast fokus. Kameran är även förberedd i Rasbian med en mängd olika alternativ och inställningar vilket förenklar hanteringen. Kamerorna är relativt ömtåliga och kan ta skada av statisk elektricitet[8].

2.1.5 IVPort

IVMech är ett turkiskt företag som arbetar med mekatronik-innovationer. De har tagit fram en modul till RPi:n som kan ta in mer än en kamera via CSI ingången. Med endast en fördröjning på 0.5 ns går dataöverföringen via kortet till RPi:n. Med en CSI ingång och fyra CSI utgångar kan simultan videoinformation tas in[7].



Figur 2 IVPort funktions layout

2.1.6 Skärm

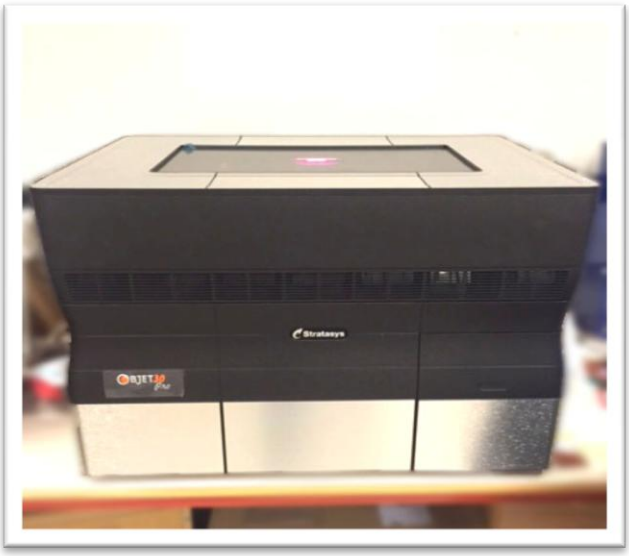
The Raspberry Pi Foundation har tagit fram en touchskärm på 7” som är kopplad via DSI. Positivt med DSI kopplingen är att HDMI porten lämnas fri vilket gör att om nödvändigt kan man köra två skärmar på RPi:n. El-överföringen i datalinorna ligger endast på 200mV vilket ger en skärm med låg energikonsumtion. Skärmen ger ifrån sig ett väldigt litet elektromagnetiskt brus, alltså nästan ingen störning[6].

2.1.7 Kabblage

Då hårdvara kopplas samman med RPi:n är det viktigt att hålla koll på strömförsörjningen. RPi:n drivs som standard av micro USB som ger 5 V med 2.4 Ampere. Om strömmen går under detta kan problem fås, vilket är en risk för om många kameror och display kopplas upp.

2.2 Object 30pro

Object 30pro är en kraftfull 3D skrivare med hög precision för att skapa realistiska modeller. Maskinen (figur 3) kan hantera olika plastmaterial som också kommer i en mängd olika färger. Det transparenta materialet veroclear är stabilt och ger fina detaljer och simulerar termoplast egenskaper så som PMMA. Även simulerat polypropylen material, RGD450 och RDG430, kan användas som ger högre styrka och hållbarhet. Maskinen använder sig av stödmaterial i gel- form, FullCore 705, som efterliknar Photopolymer stödmaterial men som inte är giftigt och mer lättanvänt. 3D skrivaren hanterar ritningar i STL-format som kan exporteras från de flesta renderingsprogram på marknaden[9].



Figur 3 Object 30pro

2.3 Mjukvara

2.3.1 Open Source

Vi har valt att arbeta med mjukvara som är open source vilket betyder att det är fritt med delning av program. Open source är ett uttryck som växer i takt med att datoranvändandet och det grundar sig i att alla ska kunna ta del av och bidra till utvecklingen av mjukvara. Källkod är något som en vardaglig datoranvändare aldrig ser eller hör något om då det oftast ändå inte går att modifiera eller göra något med den. I open source är det raka motsatsen och författarna till ett program lägger ut källkoden för alla som vill lära sig eller förbättra den och dela vidare. Att använda sig utav open source gör att det oftast är relativt enkelt att hitta information och hjälp om man stöter på problem[3].

2.3.2 OpenCV

OpenCV är ett open source bildhanteringsverktyg med många funktioner och applikationer. OpenCV skapades för att tillförse marknaden med en stabil infrastruktur inom datorseende och göra det enklare för allt från företag till privatpersoner. I biblioteket finns det mer än 2500 algoritmer som kan användas till att bland annat upptäcka ansikten, förvränga bilder med mera. Från början är openCV skrivet i programmeringsspråket C++, men med ökad efterfrågan finns det nu tillgängligt för flera olika programmeringsspråk däribland python[5].

Konfigureringen av OpenCV har sina problem och med ny hårdvara som inte är grundligt beprövad innan finns det en hel del att sätta sig in i. En mängd program och paket behövdes för att konfigureringen skulle fungera vilket gjorde att det var både plats- och tidskrävande.

2.3.3 Rasbian

Det finns en mängd olika OS som en RPi klarar av att köra, men vi har valt ett som The Raspberry Pi Foundation själva rekommenderar, Rasbian Jessie. Detta är den senaste versionen av Rasbian och bygger på det Linuxbaserade operativsystemet Debian. Som nämnt tidigare är Rasbian en del av Debian som är optimerat för att köras på en RPi. I och med att Linux är en open source mjukvara, är det många entusiaster som har varit med och bidragit till Rasbians utveckling. Jessie kommer också med en stor mängd användbara paket och förinstallerade program[4].

2.3.4 Python

Python är ett relativt ungt men kraftfullt programmeringsspråk som bygger på det mer grundliga språket C. Det enkla och rena sättet att skriva på gör att språket kan användas av allt från erfarna programmerare till nybörjare som vill testa på programmering[2]. Att ha en viss kunskap om C gör det dessutom ännu lättare att sätta sig in i och förstå Python. I Raspbian används Python som standardspråk vilket gör att man slipper installera och ta upp plats med andra språk och program då lagringen är begränsad. Kodningen kommer dessutom att ske i IDLE som är Pythons egna IDE och kommer installerat med Raspbian Jessie.

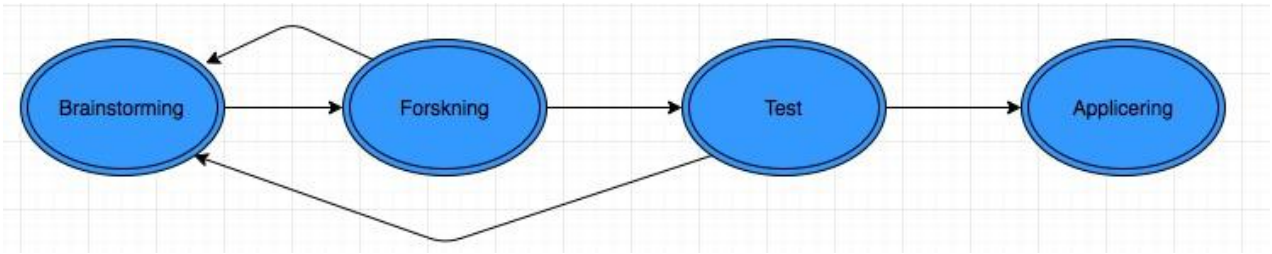
2.3.5 Catia V5

Catia är ett av de ledande 3D renderingsprogram som används inom produktutveckling. Programmet har en bred grund som täcker design, simulering och analysering. Programmet har använts för att ta fram ritningar till delar gjorda i Object 30pro skrivaren (kamerahus & skärmhållare) [10].

3 Metod

Inledningsvis togs uppgiftens tes fram genom möten och diskussioner. Projektet som valts ska röra ett område som är intressant för både företag och studenter för att nå bästa resultat. Som start gjordes en fördjupning inom området och nödvändig information söktes fram. Detta ledde till val av hårdvara som beställdes i ett tidigt stadie då vissa komponenter är ifrån utlandet och kräver viss tid för leverans. Samtidigt bekantas vi med Linux, rasbian och Raspberry Pi genom test med open source material.

Efter komplikationer med frakt av hårdvara omorganiserades tiden och fokus hamnade på kringliggande uppgifter. Sedan mer information inhämtats och alla komplikationer lösts kunde tester starta. Idéerna på kodning var många och en mängd olika program togs fram och prövades enligt metoden i figur 4.



Figur 4 Arbetsmetod

Då programutvecklingen var i gång började diskussioner om placering av kameror och dess vyer. Vi märkte snabbt att vår grundidé på placering inte skulle bli den mest optimala på grund av hårdvara så som kablage. Efter att genomförda programtester gjorts med kamerorna och ett användbart program hade tagits form, applicera detta på en bil. Att gå från en stabil miljö inne på ett kontor till en betydligt mer ostabil utomhus och gjordes inte utan sina komplikationer. Olika tester gjordes och många finjusteringar behövdes för att nå bästa resultat.

4 Hårdvarukonstruktion

4.1 Koppling iv kort

IVPort kortet löds ihop med en pin- förlängning som dels höjer kortet och ger möjlighet till vidarekoppling på RPi:ns I/O. Pinnarna 1-26 kopplar ihop korten och ger även matningsspänning som sedan regleras och fördelas på alla kameror. Det finns fyra olika inställningar för kortets pinnar. Detta görs genom att välja mellan lödhoppen A, B, C eller D (figur 4). Enligt IVMech's manual är det möjligt att genom pin konfigurering förbereda kortet för seriekoppling med lödhopp[1]. Detta är i nuläget inte aktuellt men värt att notera. IVPort sänder videodata via input och FFC till RPi:ns enda CSI ingång.

Figur 5 IVPorts lödhopp

4.2 Koppling kameror

När IVPort är ihopkopplat med RPi:n kan kameror kopplas. En enkel process för test då de endast kopplas med FFC direkt på IVPort kortet. För realisering behövs dock tester då förlängning krävs som i sin tur ger risk för störningar.

Kopplingen är först en konvertering från FFC till FRC och sen tillbaka till FFC igen för att kunna ansluta kamerorna. För att göra montering lättare på fordon kopplas även en kopplings-sko för att lätt kunna koppla ihop och ta isär en kamera åt gången. De kablarna som fungerar som förlängningen isoleras med sladdhölje och krympslang för att minska störning och städa till kabeldragningen.

Figur 6 Kabelförlängning

Enligt figur 5 förlängningen, från vänster:

IVPort → FFC → kopplingssko → FRC → kopplingssko → FFC → CAM 1/2/3/4

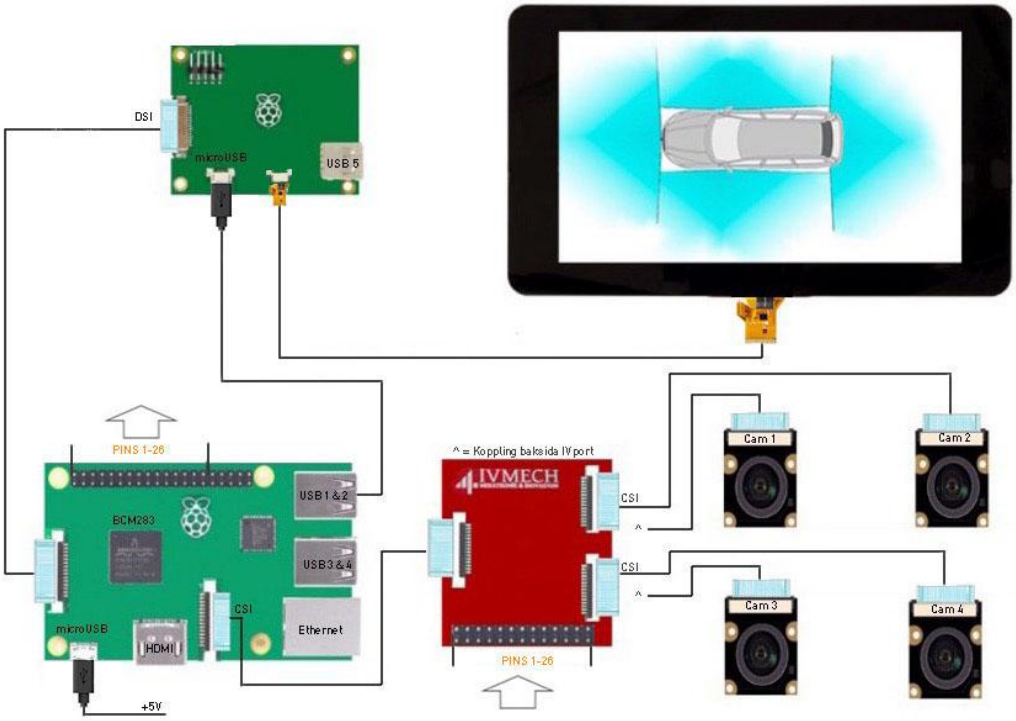
4.3 Koppling skärm

RPi:n är förberedd på att hantera externa skärmar och inköpet som vi valde var The Raspberry Pi Foundations egna touchskärm vilket gjorde allting ännu lättare. Kortet är förberett med DSI koppling och matning till skärmen kan antingen göras direkt från + 5V I/O porten och GND eller via micro-USB från Raspberry Pis fyra utgångar eller nätaggregat. Då IVMech kortet använder sig av nästan hälften av I/O-pinnarna används istället USB-matningen för att inte få störningar av delad strömfördelning.

Figur 7 Displaykoppling

4.4 Helhet

Med RPi:n som bas är nu delningskort och skärmkort stackat på varandra. Enligt figur 7 syns, med undantag av förlängningskoppling, fullständig montering av hårdvara. Stackningen av korten ger komponenterna tillräckligt med luft och minskar risken för överhettning. Vid provkodning och initiering av olika mjukvaror så kopplas touchskärmen förbi via HDMI utgången på RPi:n och en LCD-skärm används istället. För att skydda hårdvara konstrueras ett enkelt skal och stativ med monteringsfästen för delar.



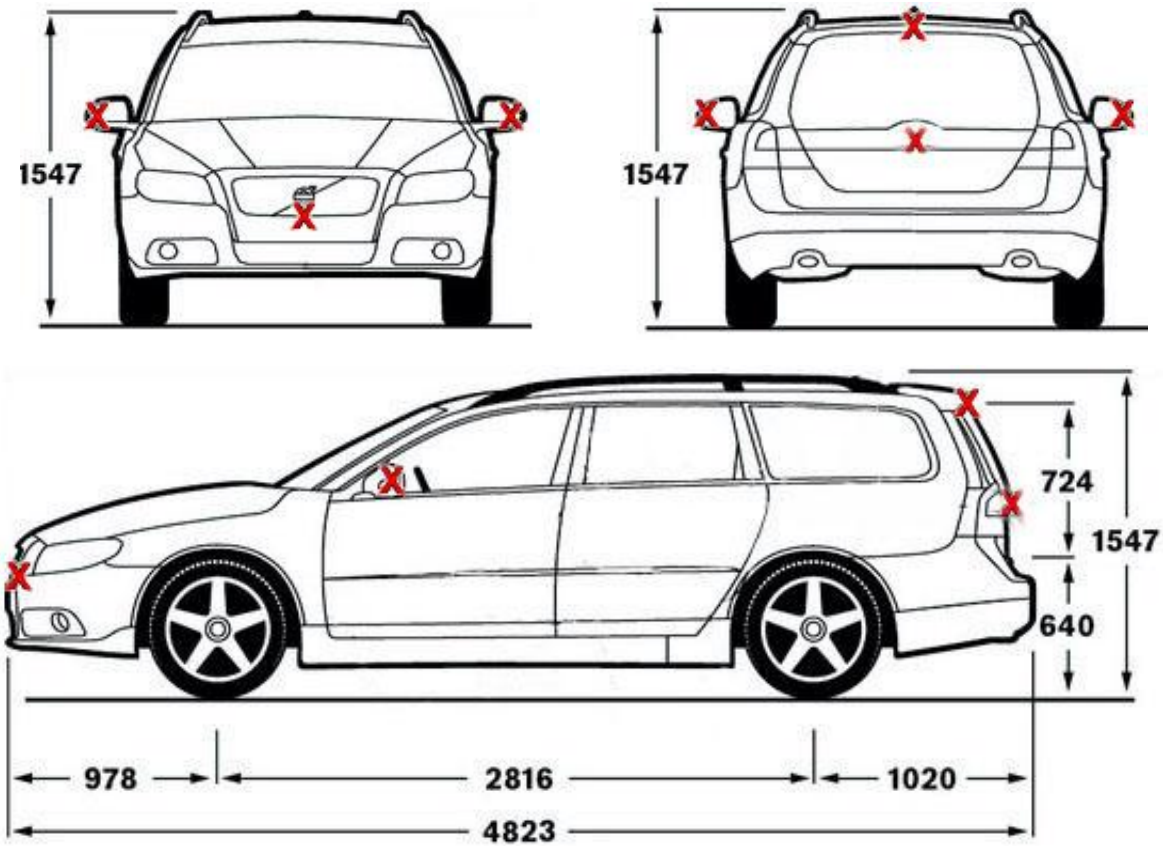
Figur 8 Komplett hårdvara

4.5 Placering

Att täcka så mycket yta som möjligt runt bilen är huvuduppgiften vid placering och sedan kommer även utseendemässiga faktorer in.

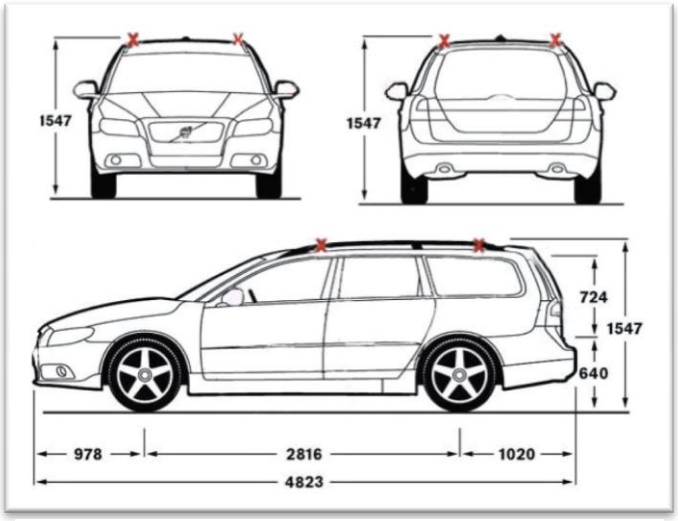
4.5.1 Placering 1

Genom diskussioner av vad som skulle fungera bäst kändes placering 1 mest naturlig med en kamera på varje sida istället för hörnfästa kameror. Backspeglar som fästpunkter gör att kamerorna kommer ut lite ifrån bilen och tar bort så mycket vy av bilen som möjligt och tar istället med omgivningen. Även höjden är viktig för hur bilden kommer tolkas med avstånd och förvrängning. Test för så verklig bild som möjligt gjordes. Förvrängningen av bilden på grund av höjdskillnaden på fram-kameran och bak-kameran visade sig vara ett problem. Lösning på detta var omplacering av bak-kamera för att få en lägre fästpunkt (figur 8).
Det diskuterades om att kooperera kamerorna rakt in i backspeglarna genom att 3D-printa plastmoduler.



Figur 9 Montage av Volvo- mått och kamerafästpunkter

4.5.2 Placering 2

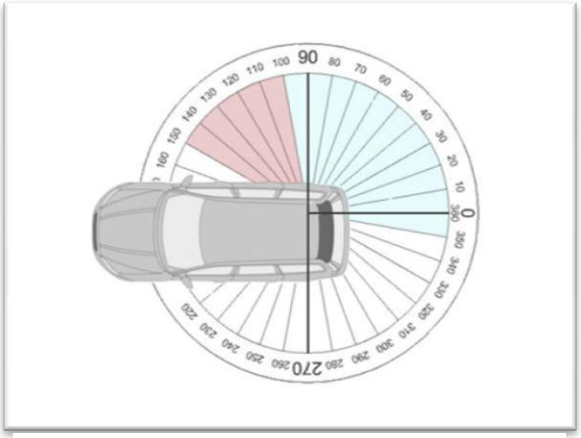


Figur 10 Montage av kamera placering 2

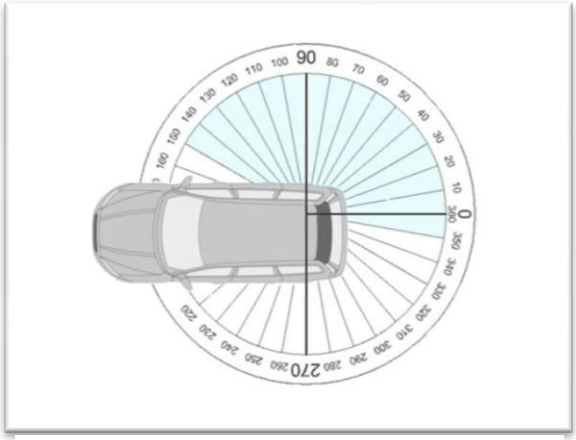
Efter att projektet tagit en vändning på grund av problemen med störningar i kablar och svårigheter med vävning av vidvinkel vyer ändrades placeringen. Med denna placering på bilen kunde kortare kablage användas och tydligare sidovyer utnyttjas. Med det nya sättet som testas täcks ca 300 grader runt bilen och verklighetsuppfattningen på längdsidorna blir betydligt bättre.
De resterande 60 graderna försvinner precis framför bilen och även direkt bakom då kamerorna inte får in tillräckligt mycket information.
Placeringen görs tillfälligt vid test och fästes provisoriskt. Vinkel på kamerorna testas också för att få så bra omfång kring bilen som möjligt. Med 4 stycken 160 graders vidvinkelkameror blir överlappen mellan kamerorna teoretiskt sätt:

160*4 = 640 Total vinkel som fås in av 4 kameror.
640 – 360 = 280 Överflödigt överlapp runt bil.
280 / 4 = 70 Delas på fyra frames.

Alltså klipps ungefär 70 graders överlapp bort på längdsidorna för att passas in med varandra.
Enligt figur 10 och 11 beskrivs överlappet som uppstår och som klipps bort.



Figur 12 Teoretiskt överlapp

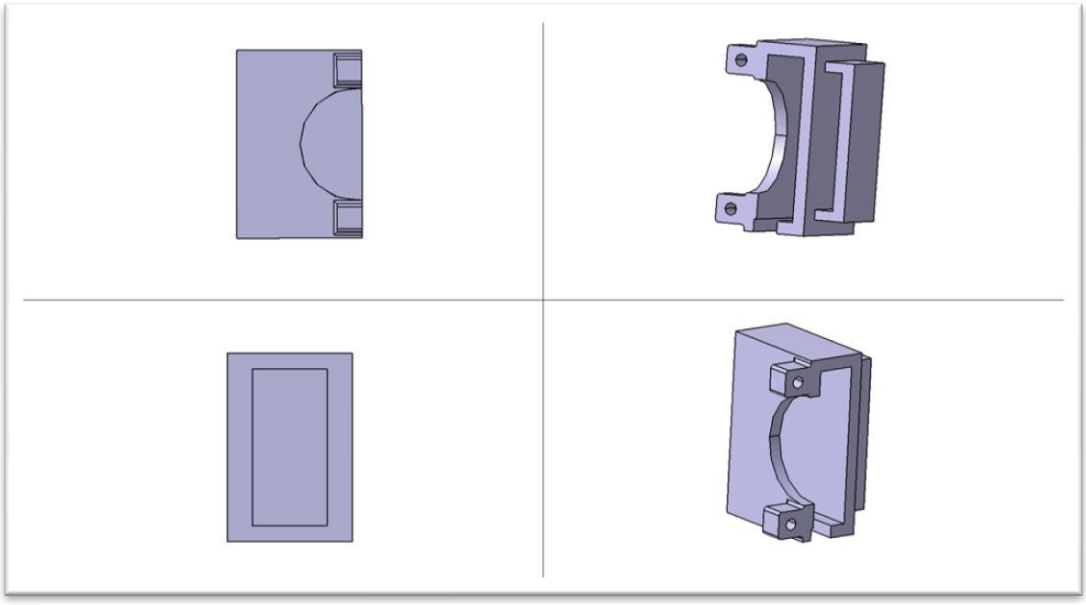


Figur 11 Vidvinkel för en kamera

I verkligheten fick passning ske via prövning även om en grund fanns för positionering. På vår provbil satt kamerorna 135 cm från backen och 135 cm ifrån varandra på långsidan. Kortsidan på taket var kamerorna placerade 160 cm ifrån varandra. Vinkel mellan kamera och marken var approximativt 35 grader och kamerornas vinkel mellan varandra på långsidan var 130 grader.

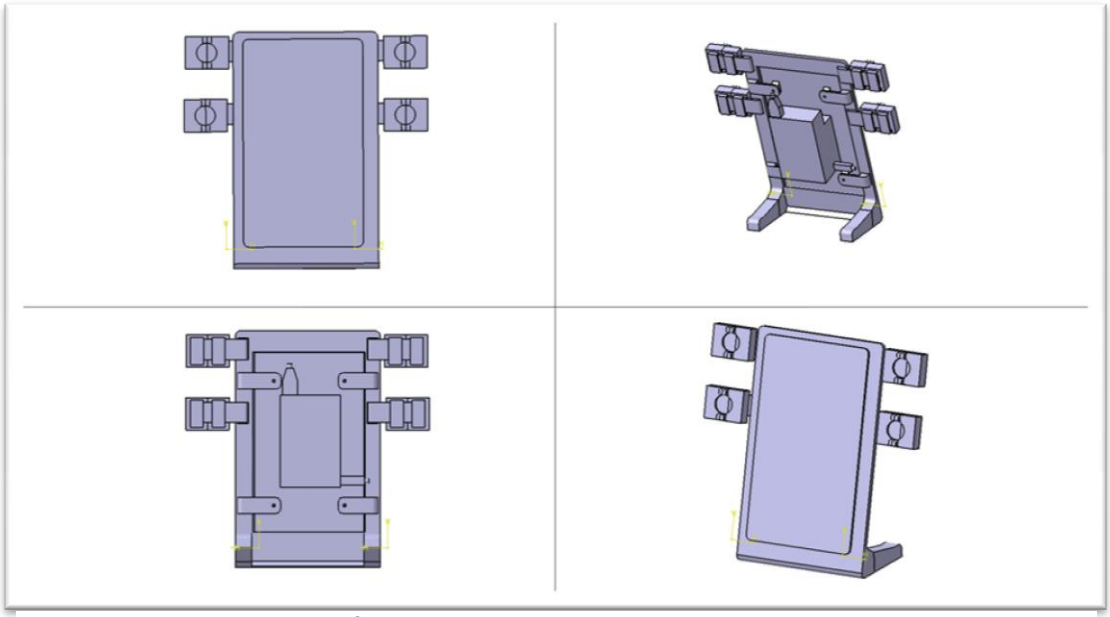
4.6 3D utskrivning

Kamerahus för skydd av de relativt känsliga Pi-kameramodulerna designades i Catia v5. Först skissades 4 olika designer upp och efter diskussioner av vad som är effektivast att skriva ut med 3D-skrivare valdes en variant med 2 delar som sätts runt kamerorna som ett skal. 3D-skrivaren som används är en Object 30pro vilket är en exklusiv maskin från Chalmers resurser. Med plastmaterialet VeroWhitePlus krävs endast 2 mm tjocka väggar för en stabil konstruktion. Enligt figur 12 ses designen som valdes. Med skruvfästen runt objektivet och fästen för upphängning på baksidan. Fyra stycken vänsterprodukter och fyra stycken höger skrevs ut.



Figur 13 Catia ritning av kamerahus, höger del

Även design för skärmhållare togs fram i Catia och efter konsultation togs en lämplig mall ut. Denna var uppdelad i fäste och fötter som monteras samman, detta för att underlätta 3D utskriften. Fästen för kameror monterades även för att underlätta hanteringen av de olika komponenterna. I figur 13 visas en monterad produkt.



Figur 14 Catia ritning displayhållare, färdigmonterad

5 Mjukvarukonstruktion

5.1 Val

Efter hårdvaruval gällde det att sätta sig in i aktuella programspråk, som nämnts innan är Python väl förberett för Raspberry Pi och även väl beprövat. Efter test och provkod insågs det att språket skiljer sig väldigt lite från C- kod och är nästan ett snyggare och framför allt ett mer lättläsligt kodspråk. Detaljer som grafisk hantering underlättas i Python och önskade programbibliotek täcker den mesta användningen i området. OpenCV valdes som tillvalsverktyg som också har ett stort bibliotek med en mängd olika applikationer.

5.2 Sd kort- plats

Som rekommendation för Raspberry Pi projekt är Sd-kort som hårdisk rekommenderat, 8 GB kort ska täcka det mesta men med tyngre verktyg som behövs för bildhantering behövdes mer utrymme. Ett 32 GB kort användes istället och detta behövdes också konfigureras då grundinställning låser kortet till endast 2 GB ledigt utrymme. Genom raspi-config inställningar kunde kortet låsas upp och utnyttjas fullt ut.

5.3 Vävning

För att väva ihop bilder och få en önskad visning av bilderna analyserades olika sätt för att lösa problemet.

5.3.1 Blending

Att lägga bilder över varandra och bestämma transparens på bestämda områden testades. På stillbild fungerade algoritmen perfekt men kunde inte appliceras som vi tänkt oss på videoformat då videoframes läggs rakt på varandra och inte överlappandes.

5.3.2 Soft edges

Att passa in videoframes bredvid varandra utan att lägga bilderna överlappandes testades också. Att sedan modifiera gränserna mellan bilderna genom att lägga ett filter som sänker skärpa på ett specifikt område med en ”blur” effekt gav inte det resultat som var väntat. Bra passning utan effekt gav bättre resultat.

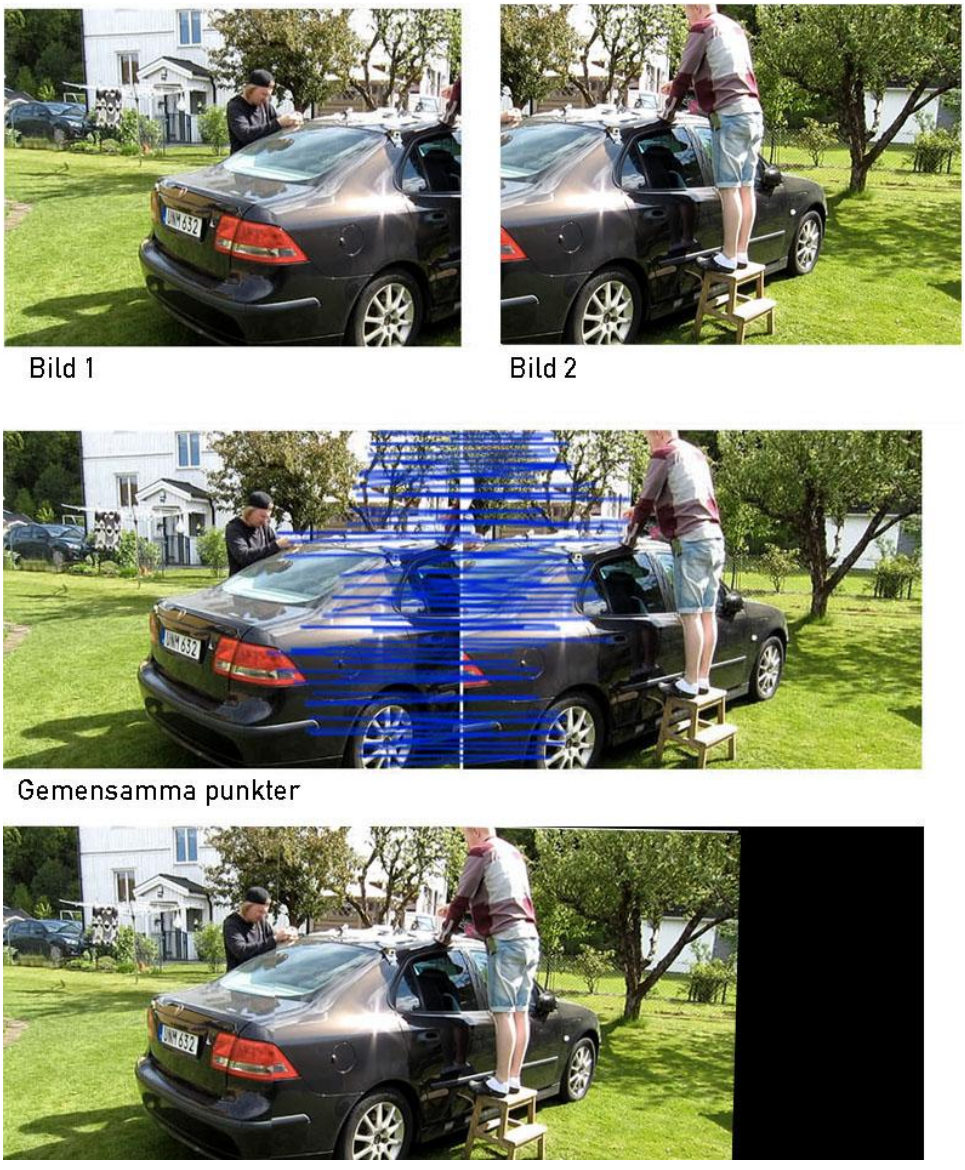
5.3.3 Placera och rotera

Då maximal Fps är önskad utan fördröjning krävdes ett sätt som var snabbt nog för hantering av den aktuella videodatan. Genom att noggrant tänka igenom placering med vinklar och avstånd insåg vi att det bästa resultatet utav det vi sökte var genom placering på bilen och enklare programmering. Genom att vrida och rotera frames och även klippa bort överflödigt överlapp kunde en realistisk bild runt bilen tas fram. Detta krävde dock mycket special anpassning för enskild montering på bil och försvårade snabb uppsättning för visning av prototyp.

5.3.4 Stitching

Ett sätt att skapa en bild ifrån två är att göra en så kallade vävning. Det kan göras på lite olika vis men ett effektivt och bra sätt är att först hitta så många punkter som möjligt som stämmer överens mellan två bilderna. När man har hittat tillräckligt med punkter använder man sig utav algoritmer för att placera bilderna på varandra och skapa en gemensam bild. Problemet med denna typ av vävning är formatet på den bild som skapas. När man väver ihop de två bilderna skapas en stor panoramabild och efter att ha vävt ihop fyra bilder får man en ännu större panorama. Formatet som vi är ute efter är 2x2 och denna typ av vävning hade gett oss 1x4 format.

Formatet var inte det enda problemet med denna typ av vävning. Kamerorna som används i projektet har fisheye- objektiv vilket gör att bilden blir förvrängd. Detta gör att punkterna som ska matchas inte stämmer överens och leder till att vävningen blir förvrängd. Lösningen på detta skulle vara att korrigera bilderna innan de vävs ihop, men den processen i sig är väldigt komplex. Enligt figur 14 ser vi processen för stitching, 2 bilder analyseras och gemensamma punkter hittas för att sedan föras samman.



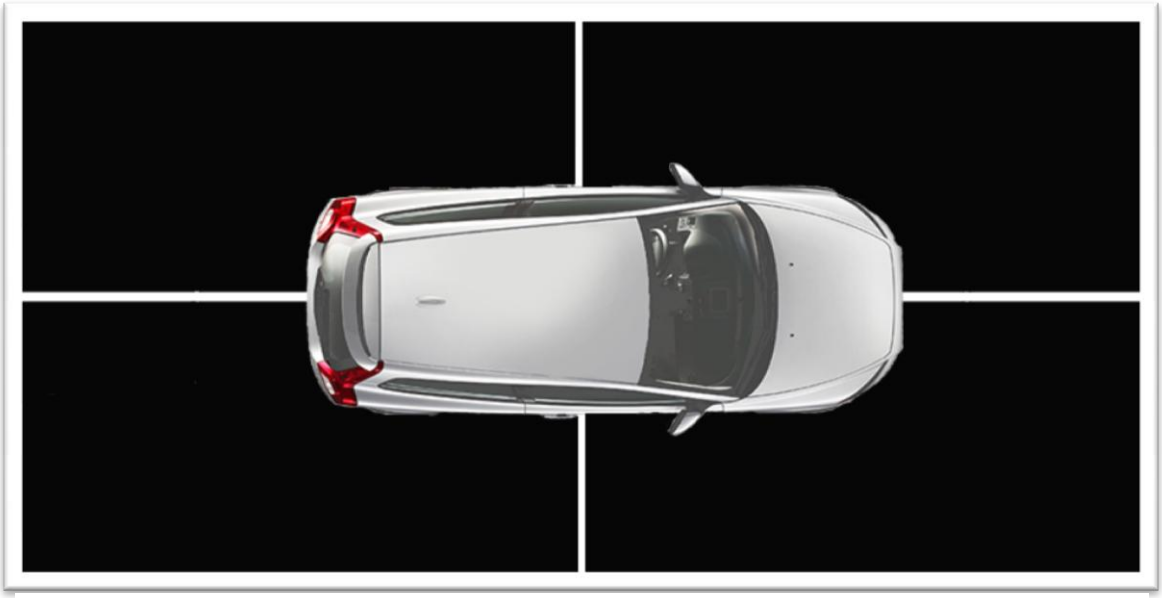
Figur 15 montage av stitching

5.4 Overlay

En av programdelarna var att få till ett lager som visas i realtid, vilket visade sig medföra olika problem.

Opencv använder sig av BGR format som standard vilket är visning av bilder med blue, green och red. Sen används också BGRA som format vilket också använder sig av en alpha kanal. Detta är för att kunna justera transparensen för bilden. Att manipulera alpha-kanalen var vårt första sätt att lösa ett overlay. Med en PNG bild öppnades kanalen upp och en transparent bild kunde läggas över realtids visning. Detta blev dock inte som vi tänkt oss då bilens mittparti var något vi ville ha helt täckt.

Istället övergick vi till en metod där vissa färger ignoreras. Oftast används grönt som i filmindustrin men svart visade sig vara bättre vid kodningen och ledde istället till enkel redigering i Photoshop för att få en önskad bild. Enligt figur 15 kan overlay bilden som använts ses. Formatet är PNG och bilden har ett rutnät som är strategiskt placerad för att täcka kanter mellan kamerabilder.



Figur 16 Volvo Overlay

5.5 Programkod

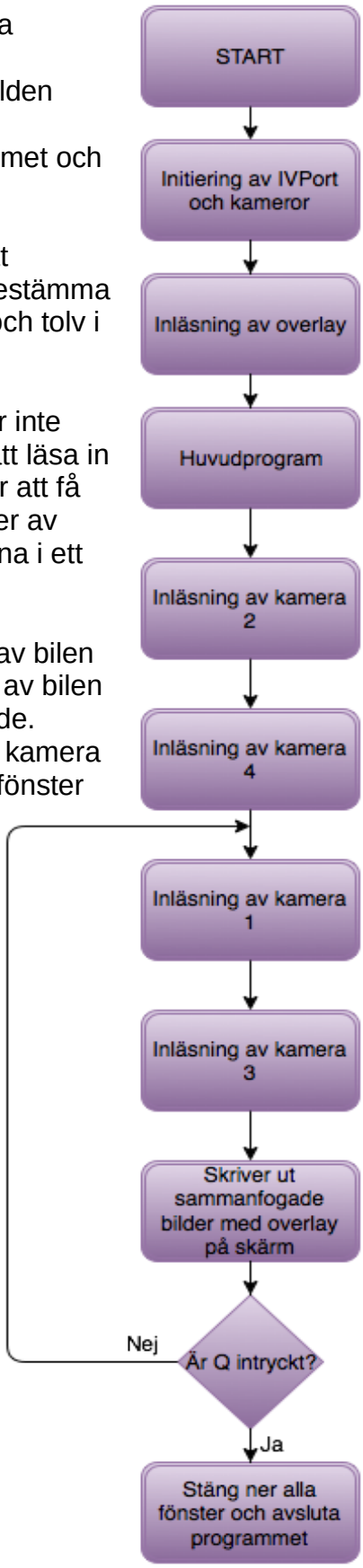
Figur 16 är en grov beskrivning av det program som kör två aktiva kameror och 2 stillbilder. Programmet börjar med initieringar av kamerorna och IVPort kortet, sedan går det vidare och läser in bilden som ska läggas ovanpå vårt videoflöde.
När alla nödvändiga förberedelser är klara startas huvudprogrammet och kamerorna börjar läsas av.

RPi:n har endast en CSI ingång vilket gjorde att vi var tvungna att använda oss utav ett IVPort kort. Detta fungerade så att vi kan bestämma vilken kamera som ska vara aktiv genom att sätta pins sju, elva och tolv i olika kombinationer.

Då vi har lite olika kodexempel för att få fram lite olika resultat ser inte alla program likadana ut. I det exempel vi har i figur 16 börjar vi att läsa in stillbilder från kamera två och fyra som beskärs och vrids 180° för att få en korrekt vy på skärmen. Programmet går sedan vidare och läser av kamera ett och tre och beskär för att sedan kunna lägga in bilderna i ett fönster.

När bilderna är inlästa och placerade på korrekt plats ska bilden av bilen läggas in. Detta gjordes genom att plocka bort allt svart ur bilden av bilen och spara allt annat och sedan lägga bilden ovanpå vårt videoflöde.
När detta är gjort fortsätter programmet att läsa in nya bilder från kamera ett och tre tills att bokstaven q trycks in på tangentbordet då alla fönster stängs ner och programmet avslutas.

Ytterligare två program gjordes för att visa upp vilken påverkan mängden aktiva kameror hade. Stommen i programmet är den samma medans växlingen mellan kamerorna skiljer sig. Ena programmet körs endast en kamera med hög fps och resterande tar stillbilder som uppdateras var femte sekund.
Det andra programmet kör alla fyra kamerorna samtidigt men med väldigt låg fps.



Figur 17 Flödesschema

6 Resultat

Med en Raspberry Pi som grund och fyra kameror kopplat via ett delningskort har ett ”proof of concept” tagits fram. En 7 tums touchskärm används för visning av den behandlade videoinformationen. Programmen som körs har olika inställningar, antingen 1,2 eller 4 aktiva kameror. Detta beroende på hur hög fps som önskas. Kamerorna placeras på bilens tak och ställs in med vinklar för att få med så mycket information som möjligt. Detta utan orealistisk förvrängning av den ihopsatta bilden. Genom att hitta rätt överlapp trimmas kamerorna in med beskärning och rotering. Realtidsvisningen från test spelas in för att kunna visas direkt på skärm utan montering på bil.

Frågeställningarna för detta projekt besvarades med blandat resultat. Att enklare hårdvara klarar att väva ihop kameror är fastställt, dock med stor fördröjning vid realtid. Med testbilder blev resultatet väldigt bra. När våra kameror användes, som har vidvinkel på 160 grader, kunde inte programmet hitta tillräckligt många gemensamma punkter att väva ihop. Bilden blev därför förvrängd.

Frågeställning två – ”Går det att visa en realistisk bild på hanterad video utan fördröjningar?” Detta ger ett delat resultat då intrimmat program blev begränsat till hastigheten på IVPort kortet som skiftar mellan kamerorna. Med två kameror i rörelse och två still ges ett resultat som fungerar för vår applikation.

Tredje och sista frågeställning – ”Är det möjligt att implementera i verklig miljö?” Detta har vi kommit fram till att det absolut gör. Tester begränsades till längd av sladdar för placering men resultatet täckte våra förväntningar.

7 Slutsats

7.1 Diskussion

Projektet har hanterat mycket forskning för att lösa våra frågeställningar och med endast lite erfarenhet sen tidigare om videohantering är det ett stort område att hitta information inom. Våra planer ändrades med tidens gång då projektet begränsades av hårdvara. Men med beslutsamhet om att ha en produkt att visa upp kom nya idéer upp för att visa vårt ”proof of concept”.

Vid starten av projektet togs ett beslut om att hålla oss till hårdvara och mjukvara som skulle vara kompatibla och fungera utan större problem med varandra. Detta gjorde att användningsområdet till viss del blev begränsat, men samtidigt gick konfigurering och att komma igång med kamerorna smidigt.

Att ha kameror som inte gick via USB var ett beslut som togs av olika anledningar så som kompatibilitet och pris. Detta kunde dock löst några utav de begränsningarna som nämnts tidigare, fps-nivå och stabilitet. Kameror med mycket vidvinkel kändes ganska naturligt då 360 graders ytor ska täckas med endast fyra kameror.

Vikten av placering av kameror och förvrängning som blir av vidvinkelkameror visade sig ha stor betydelse i denna applikation och är något som kan vidareutvecklas. För att kamerorna skulle få en så bra position som möjligt valde vi att använda FRC. Även om FFC är standard för RPi:n, tyckte vi att det var värt att konvertera till FRC. Detta gjordes för att FFC kan vara ganska svåra att hantera och om man inte har begränsat med utrymme är FRC ett bra alternativ. FRC gör även att kablarna blir mer tåliga vilket minskar risken för störningar.

Att simultant ta in fyra stycken kameror lyckades vi med IVPort kortet. Vi märkte snabbt att kortet hade vissa begränsningar och fick problem när det skulle skifta kamera i hög hastighet. Detta ledde till fördröjningar vilket hade stor inverkan på slutprodukten. Företaget IVMechs visning av produkten lovade mer än vad som kunde hållas och när deras källkod för snabbaste växling mellan kameror inte gavs ut kunde vi inte uppnå samma resultat som förväntat.

Vid valet av programmeringsspråk vägdes för- och nackdelar. Python är ett språk som växer allt mer med tiden och även om detta inte är det snabbaste språket finns det många fördelar med att välja Python. Ett starkt argument var att Python är standard språk för RPi och kommer med det OS vi använder. När efterforskning på IVPort gjordes märkte vi också att all exempelkod som gavs ut till kortet var skrivet i Python. Det ända som talade emot att använda Python var att det inte fanns någon tidigare erfarenhet av det, men då det liknar C väldigt mycket var inlärningsprocessen väldigt kort.

Diskussion om etiska aspekter har uppkommit när det kommer till filmandet av ovetande personer. Vi ser dock inte något problem med detta då ingen information lagras.

7.2 Möjligheter

Det finns en mängd olika möjligheter att förbättra och utveckla resultatet om begränsningar så som resurser och tid inte står i vägen. Beroende på vilken väg man vill att projektet ska ta kan man använda sig utav sensorer som hjälpmedel samt utveckla med ytterligare kameror för att få en bättre bild. Även om vårt resultat är ett ”proof of concept”, är det en produkt som är väldigt aktuell i dagens läge med stora företag som håller på med liknande projekt.

Referenser

[1] About us, The making of pi
https://www.raspberrypi.org/about/ (Acc 2016-03-30)

[2] The Python Tutorial, 2016-04-17
https://docs.python.org/3/tutorial/index.html (Acc 2016-04-19)

[3] What is open source, 2014-03-11
https://opensource.com/resources/what-open-source (Acc 2016-04-13)

[4] Welcome to Raspbian
https://www.raspbian.org/ (Acc 2016-04-13)

[5] About, 2013-01-24
http://opencv.org/about.html (Acc 2016-04-13)

[6] The eagerly awaited Raspberry pi display
https://www.raspberrypi.org/blog/the-eagerly-awaited-raspberry-pi-display/ (Acc 2016-04-18)

[7] IVPort manual, 2015-12-21

[8] Product description
http://www.ebay.com/itm/272175952147?_trksid=p2060353.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT (Acc 2016-04-07)

[9] Raspberry pi camera module, description,
https://www.sparkfun.com/products/11868 (Acc 216-04-07)

[9] Object 30pro
http://www.stratasys.com/3d-printers/design-series/objet30-pro (Acc 2016-05-20)

[10] Catia v5 student edition
http://academy.3ds.com/software/catia/catia-v5-student-edition/ (Acc 2016-05-20)

Bildreferenser – Alla använda bilder i denna rapport är självredigerade, självtagna eller tagna ifrån open source.

Bilagor

A Birdview_main_2camera.py

```
# -*- coding: utf-8 -*-
"""
Titel:                  Birdview_main_2camera.py
Förklaring:             Program för video/kamera hantering
Författare:             Mattias Bengtsson & Patrik Wallin
Datum:                  2016-05-22
Python version:         2.7.9
"""

#----------Import av moduler och bibliotek----------

from imutils.video.pivideostream import PiVideoStream
from imutils.video import FPS
from picamera.array import PiRGBArray
from picamera import PiCamera
import imutils
import time
import cv2
import cam_choice
import numpy as np
import RPi.GPIO as gp


#----------Initiering av IVPort----------

gp.setwarnings(False)
gp.setmode(gp.BOARD)

gp.setup(7, gp.OUT)                                     # Initierar GPIO 7, 11 och 12
gp.setup(11, gp.OUT)
gp.setup(12, gp.OUT)

gp.output(7, False)
gp.output(11, False)
gp.output(12, True)


#----------Initiering av kamera----------


camera = PiCamera()
camera.close()                                          # Stänger default aktiva kameror
cam = 2                                                 # Val av start kamera
vs = PiVideoStream().start()                            # Initierar kamera
time.sleep(1)                                           # Uppvärmning av kamera


#----------Variabler----------

frame = vs.read()
frame = imutils.resize(frame, width=400)                # Ändrar storlek på inläst frame
ratio = 500.0 / frame.shape[1]                          # Ändra dimensionen på inläst frame
dim = (400, int(frame.shape[0] * ratio))
frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)

crop1 = crop2 = crop3 = crop4 = frame                   # Lägger in start frames för att undvika error
```

```
#----------Inläsning av overlay----------------------------------------------------------------------------------------------------

overlay = cv2.imread('volvo_800_400_2.png')
ratio = 929.0 / overlay.shape[1]                                    # dimensionerar overlay
dim = (800, int(overlay.shape[0] * ratio))
overlay = cv2.resize(overlay, dim, interpolation = cv2.INTER_AREA)


#----------Huvudprogram------------------------------------------------------------------------------------------------------------

while True:


    if cam == 1:
        cam_choice.camCH(cam)                                       # Byter aktiv kamera
        time.sleep(0.13)                                            # Väntar på byte av kamera
        image1 = vs.read()                                          # Läser in bildruta
        image1 = imutils.resize(image1, width=400)

        crop1 = image1[0:300, 80:400]                               # Skalar bort överflödig information
        crop1 = imutils.resize(crop1, width=400)                    # Anpassar storlek

        cam = 3                                                     # Sätter variabel för nästa kameraval


    elif cam == 2:
        cam_choice.camCH(cam)
        time.sleep(0.15)
        image2 = vs.read()
        image2 = imutils.resize(image2, width=400)

        (h, w) = image2.shape[:2]                                   # Tar fram höjd och bredd på bildruta
        center = (w / 2, h / 2)                                     # Hittar centrum på bild
        M = cv2.getRotationMatrix2D(center, 180, 1.0)               # Skapar en rotationsmatris
        image2 = cv2.warpAffine(image2, M, (w, h))                  # Roterar bild 180 grader

        crop2 = image2[0:300, 80:400]
        crop2 = imutils.resize(crop2, width=400)

        cam = 4


    elif cam == 3:
        cam_choice.camCH(cam)
        time.sleep(0.13)
        image3 = vs.read()
        image3 = imutils.resize(image3, width=400)

        crop3 = image3[0:300, 0:320]
        crop3 = imutils.resize(crop3, width=400)

        img = np.vstack([np.hstack([crop3, crop1]), np.hstack([crop4, crop2])])        # Placerar bildrutor 2x2
        ratio = 495.0 / img.shape[1]                                # dimensionerar sammansatta frames
        dim = (800, int(img.shape[0] * ratio))
        img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

        # skapar en gråskalig kopia av vårat overlay och skapar en mask utav kopian
        overlaygray = cv2.cvtColor(overlay,cv2.COLOR_BGR2GRAY)
        ret, mask = cv2.threshold(overlaygray, 0, 255, cv2.THRESH_BINARY)
        # Gör även en inverterad kopia av masken
        mask_inv = cv2.bitwise_not(mask)
```

```
        # Gör om pixlar till svart på det område som overlay bilden ska täcka
        img_bg = cv2.bitwise_and(img,img,mask = mask_inv)

        # Väljer ut det som ska visas av overlay bilden
        overlay_fg = cv2.bitwise_and(overlay,overlay,mask = mask)
        # Lägger de två modifierade bilderna på varandra
        result = cv2.add(img_bg,overlay_fg)

        # Skapar ett fönster som täcker hela skärmen och lägger ut videoflöde
        cv2.namedWindow("Birdview", cv2.WND_PROP_FULLSCREEN)
        cv2.setWindowProperty("Birdview", cv2.WND_PROP_FULLSCREEN,
            cv2.cv.CV_WINDOW_FULLSCREEN)
        cv2.imshow("Birdview", result)

        cam = 1


    elif cam == 4:

        cam_choice.camCH(cam)
        time.sleep(0.15)
        image4 = vs.read()

        image4 = imutils.resize(image4, width=400)
        (h, w) = image4.shape[:2]
        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, 180, 1.0)
        image4 = cv2.warpAffine(image4, M, (w, h))

        crop4 = image4[0:300, 0:320]
        crop4 = imutils.resize(crop4, width=400)

        cam = 1



    key = cv2.waitKey(1) & 0xFF     # kollar om någon tangent är nedtryckt


    if key == ord("q"):                 # Om q blivit nedtryckt stängs alla fönster och kamerorna stängs av
            break

cv2.destroyAllWindows()             # Stänger alla öppna fönster
cv2.startWindowThread()
camera.close()
vs.stop()

#-------------------------------------------------------------------------------------------------------------------------------------------
```

B Birdview_main_allcamera.py

```
# -*- coding: utf-8 -*-
"""
Titel:              Birdview_main_allcamera.py
Förklaring:         Program för video/kamera hantering
Författare:         Mattias Bengtsson & Patrik Wallin
Datum:              2016-05-22
Python version:     2.7.9
"""

#----------Import av moduler och bibliotek----------

from imutils.video.pivideostream import PiVideoStream
from imutils.video import FPS
from picamera.array import PiRGBArray
from picamera import PiCamera
import imutils
import time
import cv2
import cam_choice
import numpy as np
import RPi.GPIO as gp


#----------Initiering av IVPort----------

gp.setwarnings(False)
gp.setmode(gp.BOARD)

gp.setup(7, gp.OUT)                     # Initierar GPIO 7, 11 och 12
gp.setup(11, gp.OUT)
gp.setup(12, gp.OUT)

gp.output(7, False)
gp.output(11, False)
gp.output(12, True)


#----------Initiering av kamera----------


camera = PiCamera()
camera.close()                          # Stänger default aktiva kameror
cam = 1                                 # Val av start kamera
vs = PiVideoStream().start()            # Initierar kamera
time.sleep(1)                           # Uppvärmning av kamera


#----------Variabler----------

frame = vs.read()
frame = imutils.resize(frame, width=400)        # Ändrar storlek på inläst frame
ratio = 534.0 / frame.shape[1]                  # Ändra dimensionen på inläst frame
dim = (400, int(frame.shape[0] * ratio))
frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)

crop1 = crop2 = crop3 = crop4 = frame           # Lägger in start frames för att undvika error
```

```
#----------Inläsning av overlay----------------------------------------------------------------------------------------------------

overlay = cv2.imread('volvo_800_400.png')

#----------Huvudprogram------------------------------------------------------------------------------------------------------------

while True:

    if cam == 1:
        cam_choice.camCH(cam)                                   # Byter aktiv kamera
        time.sleep(0.15)                                        # Väntar på byte av kamera
        image1 = vs.read()                                      # Läser in bildruta
        image1 = imutils.resize(image1, width=400)

        crop1 = image1[0:300, 100:400]                          # Skalar bort överflödig information
        crop1 = imutils.resize(crop1, width=400)                # Anpassar storlek

        cam = 2                                                 # Sätter variabel för nästa kameraval

    elif cam == 2:
        cam_choice.camCH(cam)
        time.sleep(0.15)
        image2 = vs.read()
        image2 = imutils.resize(image2, width=400)
        (h, w) = image2.shape[:2]                               # Tar fram höjd och bredd på bildruta
        center = (w / 2, h / 2)                                 # Tar fram centrum på bildruta
        M = cv2.getRotationMatrix2D(center, 180, 1.0)           # Skapar en rotationsmatris över bildrutan
        image2 = cv2.warpAffine(image1, M, (w, h))              # Roterar bildruta 90 grader

        crop2 = image2[0:300, 100:400]
        crop2 = imutils.resize(crop2, width=400)

        cam = 3

    elif cam == 3:
        cam_choice.camCH(cam)
        time.sleep(0.15)
        image3 = vs.read()
        image3 = imutils.resize(image3, width=400)

        crop3 = image3[0:300, 0:300]
        crop3 = imutils.resize(crop3, width=400)

        cam = 4

    elif cam == 4:

        cam_choice.camCH(cam)
        time.sleep(0.15)
        image4 = vs.read()
        image4 = imutils.resize(image4, width=400)

        (h, w) = image4.shape[:2]
        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, 180, 1.0)
        image4 = cv2.warpAffine(image4, M, (w, h))

        crop4 = image4[0:300, 0:300]
        crop4 = imutils.resize(crop4, width=400)

        cam = 1
```

```
    img = np.vstack([np.hstack([crop3, crop1]), np.hstack([crop4, crop2])])        # Placerar bildrutor 2x2
    ratio = 400.0 / img.shape[1]                                                   # dimensionerar sammansatta frames
    dim = (800, int(img.shape[0] * ratio))
    img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)



    # skapar en gråskalig kopia av vårat overlay och skapar en mask utav kopian
    overlaygray = cv2.cvtColor(overlay,cv2.COLOR_BGR2GRAY)
    ret, mask = cv2.threshold(overlaygray, 0, 255, cv2.THRESH_BINARY)
    # Gör även en inverterad kopia av masken
    mask_inv = cv2.bitwise_not(mask)

    # Gör om pixlar till svart på det område som overlay bilden ska täcka
    img_bg = cv2.bitwise_and(img,img,mask = mask_inv)

    # Väljer ut det som ska visas av overlay bilden
    overlay_fg = cv2.bitwise_and(overlay,overlay,mask = mask)
    # Lägger de två modifierade bilderna på varandra
    result = cv2.add(img_bg,overlay_fg)



    cv2.imshow("Birdview", result)

    key = cv2.waitKey(1) & 0xFF        # kollar om någon tangent är nedtryckt



    if key == ord("q"):                # Om q blivit nedtryckt stängs alla fönster och kamerorna stängs av
        break

cv2.destroyAllWindows()                # Stänger alla öppna fönster
cv2.startWindowThread()
camera.close()
vs.stop()


#-------------------------------------------------------------------------------------------------------------------------------
```

C Birdview_main.py

```
# -*- coding: utf-8 -*-
"""
Titel:                  Birdview_main.py
Förklaring:             Program för video/kamera hantering
Författare:             Mattias Bengtsson & Patrik Wallin
Datum:                  2016-05-23
Python version:         2.7.9
"""

#----------Import av moduler och bibliotek----------

from imutils.video.pivideostream import PiVideoStream
from imutils.video import FPS
from picamera.array import PiRGBArray
from picamera import PiCamera
import imutils
import time
import cv2
import cam_choice
import numpy as np
import RPi.GPIO as gp


#----------Initiering av IVPort----------

gp.setwarnings(False)
gp.setmode(gp.BOARD)

gp.setup(7, gp.OUT)                                     # Initierar GPIO 7, 11 och 12
gp.setup(11, gp.OUT)
gp.setup(12, gp.OUT)

gp.output(7, False)
gp.output(11, False)
gp.output(12, True)


#----------Initiering av kamera----------


camera = PiCamera()
camera.close()                                          # Stänger default aktiva kameror
cam = 2                                                 # Val av start kamera
vs = PiVideoStream().start()                            # Initierar kamera

time.sleep(1)                                           # Uppvärmning av kamera


#----------Variabler----------

t_end  = 1
frame = vs.read()
frame = imutils.resize(frame, width=400)                # Ändrar storlek på inläst frame
ratio = 500.0 / frame.shape[1]                          # Ändra dimensionen på inläst frame
dim = (400, int(frame.shape[0] * ratio))
frame = cv2.resize(frame, dim, interpolation = cv2.INTER_AREA)
crop1 = crop2 = crop3 = crop4 = frame                   # Lägger in start frames för att undvika error
```

```
#----------Inläsning av overlay----------------------------------------------------------------------------------------------------

overlay = cv2.imread('volvo_800_400_2.png')
ratio = 929.0 / overlay.shape[1]                                    # dimensionerar overlay
dim = (800, int(overlay.shape[0] * ratio))
overlay = cv2.resize(overlay, dim, interpolation = cv2.INTER_AREA)


#----------Huvudprogram------------------------------------------------------------------------------------------------------------

while True:


    if cam == 1:

        cam_choice.camCH(cam)                                       # Byter aktiv kamera
        time.sleep(0.10)                                            # Väntar på byte av kamera
        image1 = vs.read()                                          # Läser in bildruta
        image1 = imutils.resize(image1, width=400)

        crop1 = image1[0:300, 80:400]                               # Skalar bort överflödig information
        crop1 = imutils.resize(crop1, width=400)                    # Anpassar storlek

        # Placerar bildrutor 2x2
        img = np.vstack([np.hstack([crop3, crop1]), np.hstack([crop4, crop2])])
        ratio = 495.0 / img.shape[1]                                # dimensionerar overlay
        dim = (800, int(img.shape[0] * ratio))
        img = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

        # skapar en gråskalig kopia av vårat overlay och skapar en mask utav kopian
        overlaygray = cv2.cvtColor(overlay,cv2.COLOR_BGR2GRAY)
        ret, mask = cv2.threshold(overlaygray, 0, 255, cv2.THRESH_BINARY)
        # Gör även en inverterad kopia av masken
        mask_inv = cv2.bitwise_not(mask)

        # Gör om pixlar till svart på det område som overlay bilden ska täcka
        img_bg = cv2.bitwise_and(img,img,mask = mask_inv)

        # Väljer ut det som ska visas av overlay bilden
        overlay_fg = cv2.bitwise_and(overlay,overlay,mask = mask)
        # Lägger de två modifierade bilderna på varandra
        result = cv2.add(img_bg,overlay_fg)

        # Skapar ett fönster som täcker hela skärmen och lägger ut videoflöde
        cv2.namedWindow("Birdview", cv2.WND_PROP_FULLSCREEN)
        cv2.setWindowProperty("Birdview", cv2.WND_PROP_FULLSCREEN,
            cv2.cv.CV_WINDOW_FULLSCREEN)
        cv2.imshow("Birdview", result)


    elif cam == 2:
        cam_choice.camCH(cam)
        time.sleep(0.10)
        image2 = vs.read()
        image2 = imutils.resize(image2, width=400)

        (h, w) = image2.shape[:2]                                   # Tar fram höjd och bredd på bildruta
        center = (w / 2, h / 2)                                     # Hittar centrum på bild
        M = cv2.getRotationMatrix2D(center, 180, 1.0)               # Skapar en rotationsmatris
        image2 = cv2.warpAffine(image2, M, (w, h))                  # Roterar bild 180 grader
```

```
        crop2 = image2[0:300, 80:400]
        crop2 = imutils.resize(crop2, width=400)

        cam = 3


    elif cam == 3:
        cam_choice.camCH(cam)
        time.sleep(0.10)
        image3 = vs.read()
        image3 = imutils.resize(image3, width=400)

        crop3 = image3[0:300, 0:320]
        crop3 = imutils.resize(crop3, width=400)

        cam = 4


    elif cam == 4:

        cam_choice.camCH(cam)
        time.sleep(0.10)
        image4 = vs.read()
        image4 = imutils.resize(image4, width=400)

        (h, w) = image4.shape[:2]                    # Tar fram höjd och bredd på bildruta
        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, 180, 1.0)
        image4 = cv2.warpAffine(image4, M, (w, h))

        crop4 = image4[0:300, 0:320]
        crop4 = imutils.resize(crop4, width=400)

        cam = 1


    if time.time() > t_end:          # Kör kamera 1 i 5 sekunder för att sedan uppdatera resterande kameror
        if cam == 1:
            cam = 2
            t_end = time.time() + 5

    key = cv2.waitKey(1) & 0xFF  # kollar om någon tangent är nedtryckt


    if key == ord("q"):              # Om q blivit nedtryckt stängs alla fönster och kamerorna stängs av
            break

cv2.destroyAllWindows()          # Stänger alla öppna fönster
cv2.startWindowThread()
camera.close()
vs.stop()

#-------------------------------------------------------------------------------------------------------------------------
```

D cam_choice.py

```
# -*- coding: utf-8 -*-

"""
Titel:                  cam_choice.py
Förklaring:             Program för kamera byte
Författare:             Mattias Bengtsson & Patrik Wallin
Datum:                  2016-05-16
Anteckningar:           Lödning IVPort A-inställningar
Python version:         2.7.9
"""

import RPi.GPIO as gp
import time

def camCH(cam):

    if cam == 1:                    # Aktiverar kamera 1
        gp.output(7, False)
        gp.output(11, False)
        gp.output(12, True)


    elif cam == 2:                  # Aktiverar kamera 2
        gp.output(7, True)
        gp.output(11, False)
        gp.output(12, True)


    elif cam == 3:                  # Aktiverar kamera 3
        gp.output(7, False)
        gp.output(11, True)
        gp.output(12, False)


    elif cam == 4:                  # Aktiverar kamera 4
        gp.output(7, True)
        gp.output(11, True)
        gp.output(12, False)
```

E FPS.py

```
# -*- coding: utf-8 -*-
"""
Titel:                  FPS.py
Förklaring:             Program för att räkna fps
Författare:             Adrian Rosebrock
Datum:                  2016-05-16
Python version:         2.7.9
"""
import datetime

class FPS:
        def __init__(self):
                # store the start time, end time, and total number of frames
                # that were examined between the start and end intervals
                self._start = None
                self._end = None
                self._numFrames = 0

        def start(self):
                # start the timer
                self._start = datetime.datetime.now()
                return self

        def stop(self):
                # stop the timer
                self._end = datetime.datetime.now()

        def update(self):
                # increment the total number of frames examined during the
                # start and end intervals
                self._numFrames += 1

        def elapsed(self):
                # return the total number of seconds between the start and
                # end interval
                return (self._end - self._start).total_seconds()

        def fps(self):
                # compute the (approximate) frames per second
                return self._numFrames / self.elapsed()
```

F PiVideoStream.py

```
# -*- coding: utf-8 -*-
"""
Titel:              PiVideoStream.py
Förklaring:         Program för att öka fps
Författare:         Adrian Rosebrock
Datum:              2016-05-16
Python version:     2.7.9
"""

from picamera.array import PiRGBArray
from picamera import PiCamera
from threading import Thread
import cv2

class PiVideoStream:
	def __init__(self, resolution=(800, 480), framerate=32):
		# initialize the camera and stream
		self.camera = PiCamera()
		self.camera.resolution = resolution
		self.camera.framerate = framerate
		self.rawCapture = PiRGBArray(self.camera, size=resolution)
		self.stream = self.camera.capture_continuous(self.rawCapture,
			format="bgr", use_video_port=True)

		# initialize the frame and the variable used to indicate
		# if the thread should be stopped
		self.frame = None
		self.stopped = False

	def start(self):
		# start the thread to read frames from the video stream
		Thread(target=self.update, args=()).start()
		return self

	def update(self):
		# keep looping infinitely until the thread is stopped
		for f in self.stream:
			# grab the frame from the stream and clear the stream in
			# preparation for the next frame
			self.frame = f.array
			self.rawCapture.truncate(0)

			# if the thread indicator variable is set, stop the thread
			# and resource camera resources
			if self.stopped:
				self.stream.close()
				self.rawCapture.close()
				self.camera.close()
				return

	def read(self):
		# return the frame most recently read
		return self.frame

	def stop(self):
		# indicate that the thread should be stopped
		self.stopped = True
```