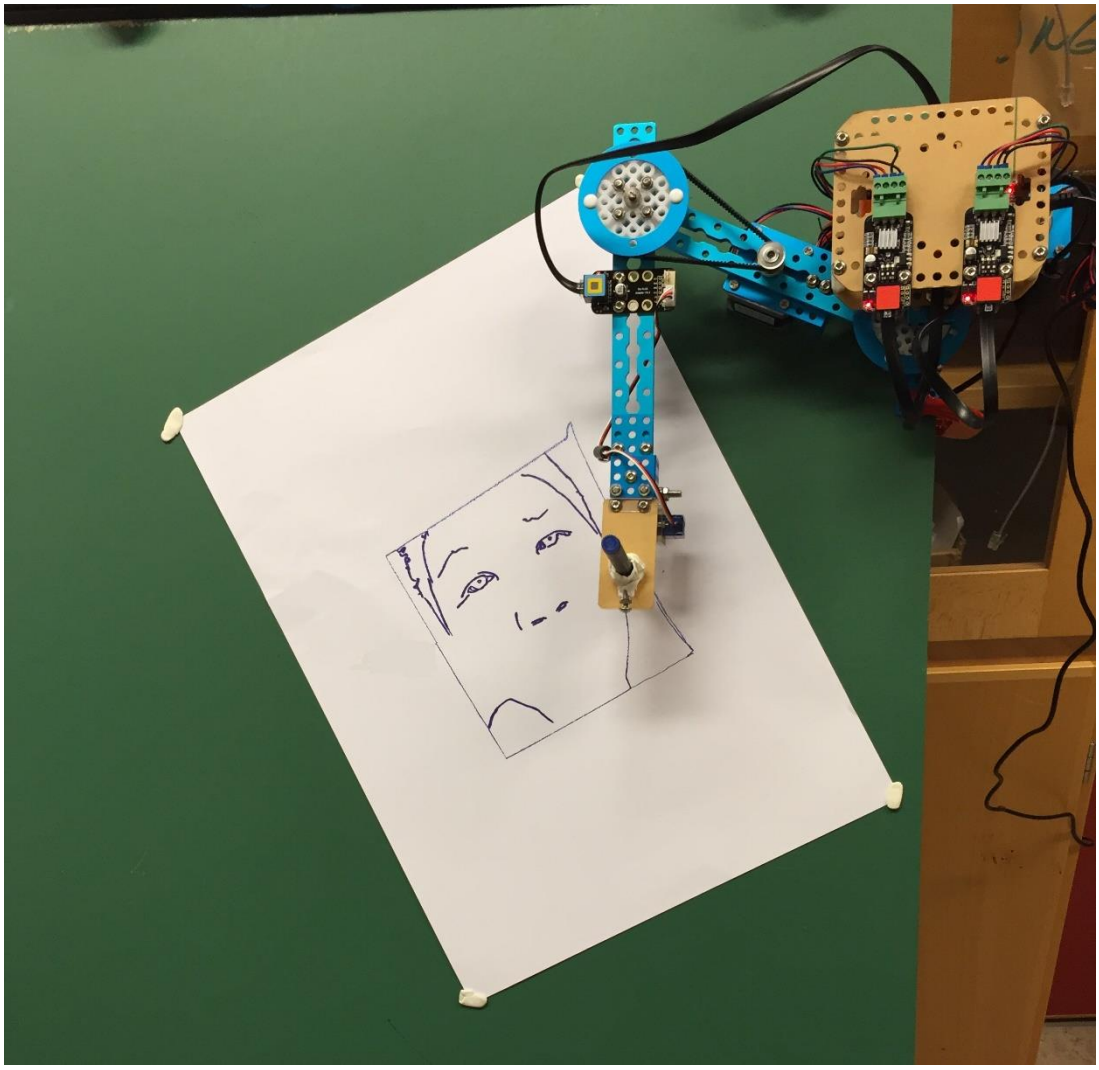
CHALMERS



Konvertera Pixlar till Linjer

Examensarbete inom Data- och Informationsteknik

Erik Samaha
Min Cheng

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

EXAMENSARBETE

Konvertera Pixlar till Linjer

En analys kring bildbehandling

Med den utrustning och de krav som utformats kan en optimal behandling av en bild från pixlar till linjer framkomma.

Erik Samaha
Min Cheng

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg 2017

Konvertera Pixlar till Linjer
En analys kring bildbehandling
Erik Samaha
Min Cheng

Examinator: Peter Lundin

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: Illustration av den framtagna anordningen i utritning av ett porträtt

Institutionen för Data- och Informationsteknik
Göteborg 2017

SAMMANFATTNING

Arbetet utfördes för Chalmers Teknisk Högskola, med bakgrunden att utveckla en robot som utifrån en bild kan rita ett porträtt av personer eller grupp av personer. Raspberry Pi används som plattform för hårdvaran och mjukvaran är utvecklad med språket Python. Det mest väsentliga problemet har varit att konvertera ursprungsbilden till ett format som tar bort bakgrund, färger och nyanser samt framhäver konturer. Iden bakom detta arbete är framtagen av Sakib Sistek från Chalmers Tekniska Högskola.

Inom ramen för projektet har det utvecklats ett system som kan fotografera ett ansikte och efter bildbehandling via en robotarm rita en skiss av ansiktet. I det slutliga systemet användes biblioteket OpenCv för att omvandla en bild från pixlar till linjer. Valet av OpenCv är baserat på en analys av befintliga program som är öppen källkod.

Operativsystemet som installerats på Raspberry Pi är Linux baserat. Robotarmen har lyckats att rita ut de linjäriserade bilder som laddats till den. Skisserna som systemet har ritat blev tillräckligt tydliga för att identifiera personen som fotats med hjälp av en till systemet ansluten Pi-kamera. Inom projektet har även operativsystemet Windows 10 IoT Core testats men det visade sig inte fungera för detta projekts ändamål.

Robotarmen som använts under projektet är ett förbeställt kit med egen mjukvara. All mjukvara som använts under arbetet öppen källkod.

Nyckelord: Python, Rasperry Pi, Pi-kamera, Bildbehandling, robotarm

ABSTRACT

This thesis was performed at Chalmers University of Technology, with the background of developing a robot that, based on an image, can draw a portrait of an individual or a group of people. Raspberry Pi is used as a platform for hardware and software which is developed with the programming language Python. The most essential problem has been to convert the original image into a format that deletes backgrounds, colors and shades, and highlights contours. The idea behind this thesis was developed by Sakib Sistek from Chalmers University of Technology.

Within this project, a system has been developed able to photograph a face and after image processing draw a sketch of the face with a robot arm. In the final system, OpenCV library was used to convert an image from pixels to lines. The choice of OpenCv is based on an analysis of existing open source programs for image processing.

The operating system installed on the Raspberry Pi is based on Linux. The robot arm has managed to draw out the linearized images loaded to it. The sketches drawn by the system became adequately clear enough to identify the person photographed using the Pi-camera connected to the system. Within the project, the Windows 10 IoT Core operating system has been tested, but it did not wort for the purpose of this project.

The robot arm used during the project is a pre-ordered kit with its own software. All software used during the work is open source.

Keywords: Python, Raspberry Pi, Pi-camera, image processing, robot arm

FÖRORD

Detta examensarbete är det sista moment på högskoleingenjörsprogram mekatronik (180 hp) vid Chalmers tekniska högskola, arbete utfördes under 20 veckor på vårterminen 2017 på halvtid.

Tack våra familjer för att ni fött och uppfostra oss, låtit oss bli goda samhällsmedborgare.

Sist men inte minst vill vi tacka vår handledare Sakib Sistek, som gav oss råder och tips, guida oss när vi stötte på problem, men som störst gav oss möjligheten att utföra detta projekt som vårt examensarbete. Vi vill också tacka Peter Lundin som var examinator.

Innehållsförteckning

1 Inledning
Introduktionen presenterar ämnet som detta examensarbete behandlar samt vad som försöks att uppnås i slutet av arbetet med de begränsningar och krav som ställs.

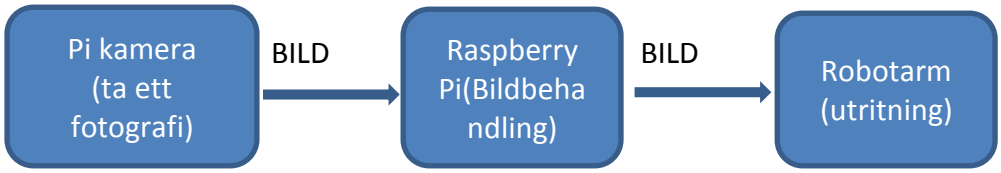
1.1. Bakgrund
Inom arbetets ram ska en robot utvecklas som kan rita ett porträtt av personer eller grupp av personer. Roboten ska användas vid olika Chalmersevenemang, där man ska erbjuda gäster att få sitt porträtt utritat. Planerad visning är vid Chalmersdagarna i Runan på Johanneberg samt biblioteket i Kuggen på Lindholmen, alternativt liknande informations och rekryteringsdagar. Iden bakom arbete är framtagen av Sakib Sistek från Chalmers Tekniska Högskola.

1.2. Syfte
Syftet med detta projekt är att utveckla ett system som kan ta en bild, konvertera bilden samt rita ut bilden med hjälp av en robotarm. Systemets bildbehandlingsprogram ska omvandla en bild från pixlar till linjer, så att tillräckligt av bildens ursprungliga information finns kvar för att kunna identifiera objekten i bilden. Målet med projekt är även att robotarmen ska klara av att rita en linjäriserad bild.

1.3. Mål
Målet med detta projekt är att med hjälp av ett datorkort Raspberry Pi och en robotarm utveckla ett system som kan skissa en bild från en bild. Systemet ska inkludera ett bildbehandlingsprogram som kan konvertera en bild tagen med hjälp av systemets Pi-kamera, så att bildens konturer kan ritas av en robotarm, (se figur 1.1).

1.4. frågeställning
Frågor som ska besvaras i slutet av arbetet:

- Vilket av de operativsystem som går att installera på en Raspberry Pi ger optimalt resultat?
- Vilken programvara eller bibliotek till ger mest optimal bildbehandling resultat?
- Vilken position för kamera och avstånd till individen som fotas är acceptabel för att ritat ansikte som efterliknar personen.



Figur 1.1: *Ett överskådligt blockschema på utseendet av programmet och bildens väg igenom.*

1.5. Avgränsningar
Robotarmen som ska användas under projektet är ett färdigt kit med egen mjukvara. All mjukvara som används under arbetet ska vara öppen källkod. Plattformen Raspberry Pi är bestämt i förväg av uppdragsgivaren.

2. Teknisk Bakgrund

Under utförandet av detta projekt användes ett bestämt programmeringsspråk, operativsystem, plattform samt specifika programvaror. Dessutom användes ett färdigt kit för uppbyggnaden av robotarmen och kunskap i bildbehandling.

2.1 Mjukvara

I projektet har följande programvaror använts.

2.1.1 Python

Python är en högnivå programmeringsspråk som är ett allmänt programutvecklingsspråk [1], som är designat för att utveckla mjukvara till ett brett applikationsområde [2]. Språket utformades av Guido van Rossum under 1980-talet och släpptes år 1991. Idén bakom Python är att betona kodens läsbarhet och syntax vilket gör det möjligt för programmerare att utveckla mjukvara med färre kodrader jämfört med andra programmeringsspråk [1] T.ex. C++ och Java (tusen rader kod i Python kan motsvara cirka 10 tusen rader i C). Språk ger den möjligheten att skriva program både i stor och lite skala.

Python stödjer flera Programmeringsparadigm (teori och grundläggande arbetssätt för hur programmet ska organiseras och struktureras) såsom objektorienterad, funktionell, procedurell programmering [4]. Det innehåller också ett stort och omfattande standardbibliotek. Eftersom Python kompilatorer finns tillgängligt på många operativsystem kan språket användas för en mängd olika system.

2.1.2 PyQt

Ett bibliotek för Python som användes i projektet är PyQt, beroende på att programmet för robotarmen är kodad med hjälp av detta bibliotek. PyQt gör det möjligt att använda Qt, en multiplattforms applikationsutvecklings ram [6]. Qt är inte ett programmeringsspråk i sig, utan ett bibliotek skrivet i C++. Som innehåller många moduler för exempelvis grafik, audio, video etc. I projekt används specifikt Qt GUI för att skapa ett användarinterface för programmet för robotarmen.

Qt startades att utvecklas under 1990-talet av två norska programmerare, Eirik Chambe-Eng och Haavard Nord. Deras idé vare att skapa ett objektorienterat grafiskt användargränssnitt, vilket blev framtagandet av grunden för dagens plattformsoberoende grafiska användargränssnitt (metod för interaktion mellan individ och dator) [7]. Första lanseringen var 20 maj 1995.

2.1.3 OpenCv(Open Source Computer Vision)

Ett annat viktigt bibliotek som är anpassat för bland annat Python är *OpenCv*, vilket har använts i projektet som ett alternativ vid bildbehandling. OpenCv består av programmerings funktioner avsedda för att hämta in, processa, analysera och förstå digitala bilder [8]. Ursprungligen utvecklat år 1999 av Intels utvecklingscentrum i Ryssland. År 2012 tog företag Itseez över, de upprätthåller utveckling av OpenCV och dess supportsidan [9], för att efteråt 26 maj 2016 köpas upp av Intel [10]. Det är dock fortfarande Itseez som tar hand om allting relaterat till OpenCv.

Biblioteket finns för C, C++, Python och Java samt kan användas på operativsystem såsom Windows, Linux, Android etc. Biblioteket innehåller över 3000 funktioner.

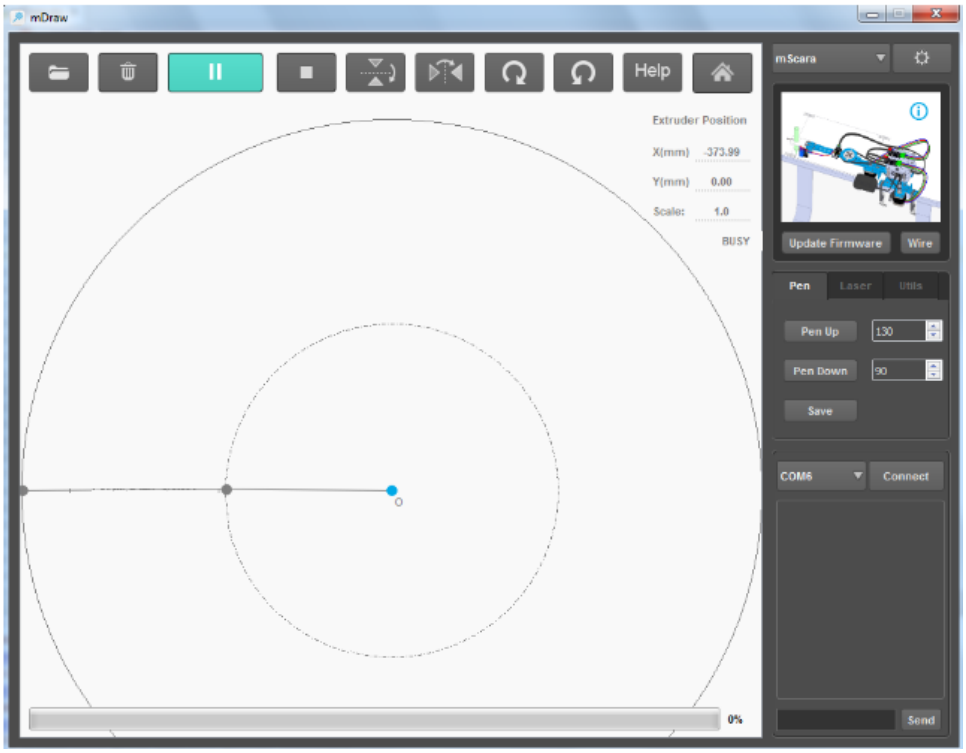
2.1.4 Potrace

Potrace är ett bibliotek som tillhöra Python [14]. Dess funktion är att transformera en bild i ett filformat till ett annat filformat. Biblioteket stödjer enbart *bitmap*'s som indata vilket finns i formaten PBM, PGM, PPM, och BMP. Utdata som den kan transformera indata till är också begränsad till ett antal format. I projektet användes Potrace till att omvandla en bild i filtyp format BMP till en vektorgrafik filtyp som programmet för robotarmen kan läsa, och rita ut.

Bitmap definieras som ett nätverk av simpla rektanglar bestående av pixlar, där varje enskild pixel lagra ett färgvärde [20]. Bitmap karaktäriseras av två parametra. Antalet pixlar och färg i form av visst antal bitar per pixel. Det finns också andra kännetecken som varierar från olika format men de är baserade på de tidigare två.

2.1.5 mDraw

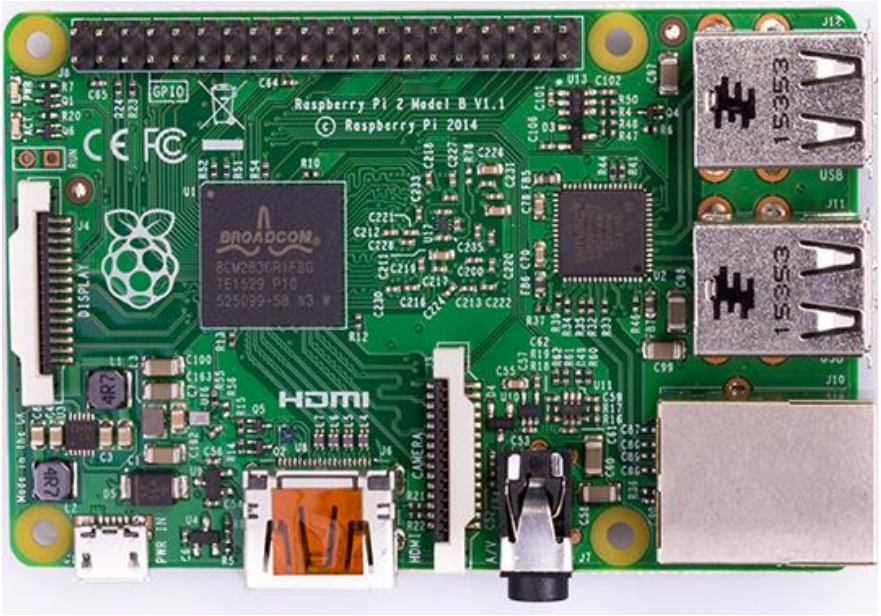
Robotarmen som beställdes innehöll också en färdig programvara som heter mDraw. mDraw är i dagsläget ett Windows baserat program, men kommer att finns tillgänglig för Mac OS X och Linux. Programvaran är utvecklad så att ett fönster dyker upp på datorskärmen varje gång man vill starta robotarmen, se figur 2.8.1. Fönstret kräver att man klickar på knappar för att välja olika alternativ, exempelvis USB-port, hastighet, placeringen på pappret som bilden ska ritas på etc. Detsamma gäller val av bild som ska ritas, allt detta utförs med knapptryck. Men då det är valt i detta arbete att alla funktioner i mDraw ska utföras utan knapptryck, har det krävts att koden för programvaran modifierats. Programvaran mDraw används i systemet som utvecklats under projektet, systemet är Linux baserat vilket medför att vissa begräsningar av funktioner finns.



Figur 2.8.1: *En bild på programmet mDraw öppnat i ett dator fönster, en självtagen bild*

2.2. Hårdvara
Vidare beskrivs den hårdvaran som använts i projektet.

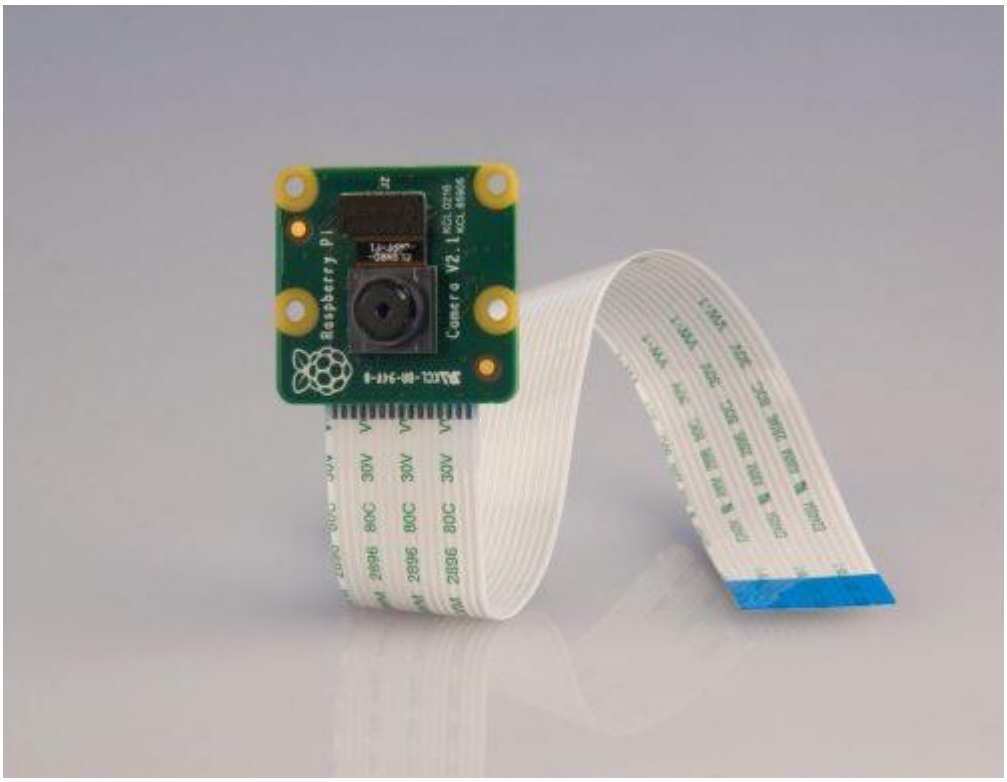
2.2.1 Raspberry Pi
Raspberry Pi är ett kreditkorts storleks enchipsdator som är utvecklad i Storbritannien av Raspberry Pi Foundation [3], i syfte att bidra till undervisningen av grundläggande datakunskap i skolan. Det är en kraftfull minidator som kan användas inom många olika områden och kan principiellt utföra likartade funktioner som en persondator. Raspberry Pi har normalt ett operativsystem, Linux, vilket innebär att flera program kan exekveras parallellt. Jämfört med andra plattformar när man vill utforma egna program för olika områden är det betydligt mer komplicerat att göra det på Raspberry Pi, men Raspberry har förmåga och högre kompatibilitet att utföra mer avancerade uppdrag/arbeten.



Figur 2.5: *En bild på Raspberry Pi 2 modell B, hämtad från Raspberry Pi produkt hemsida [21].*

2.2.2 Raspberry Pi Camera Board v2.1

I detta projekt användes en kamera som är designad och konstruerad av Raspberry Pi Foundation. Kretskort självt är ganska lite, och väger omkring 3g, vilket gör den perfekt för mobila eller andra tillämpningar [13]. Den kamera som användes har storlek 25mm x 23mm x 9mm och upplösningen 8 megapixlar, kameran kan ta bilder med formatet 3280 x 2464. Det är kompatibel med Raspberry Pi 1,2 och 3, har stöd för 1080p/30fps, 720p/60fps och 640x480p/90fps video. Jämfört med de tidigare generationerna av Pi-kamera, har version 2.1 bättre bildkvalitet och färgåtergivning.



Figur 2.6: *En bild på Raspberry Pi Camera Board v2.1, hämtad ur en artikel från en hemsida tillhörande Raspberry Pi Foundation [13].*

2.2.3 PiTFT Plus (Touchscreen)

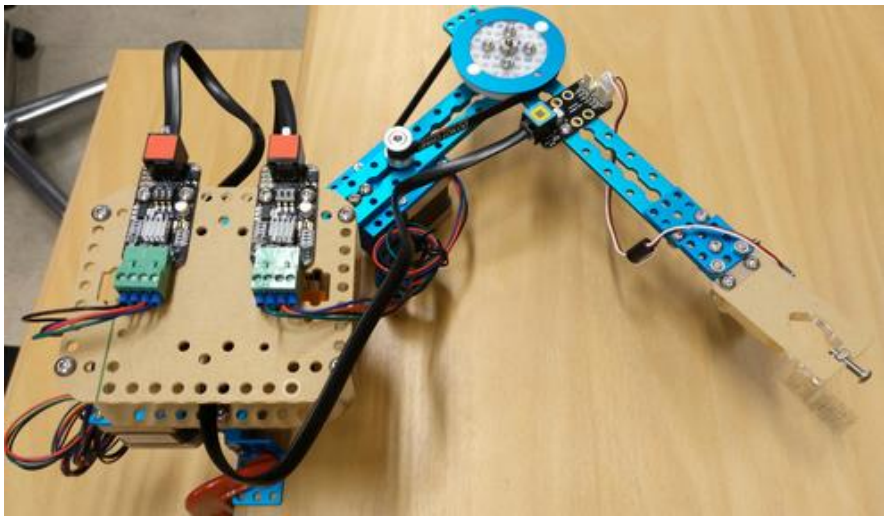
PiTFT Plus är en TFT pekskärm som har storlek 2.8", upplösning är 320x240 med kapacitiv pekskärm, se figur 2.7, vilken betyder att den stödjer multi touch funktion med upp till högst 10 fingerar [17]. Skärmen passar perfekt till Raspberry Pi 2 och 3 modell A+, B+, som har 2x20 stift.

I projektet används displayen för att tillåta användaren se ansikte när ett fotografi tagits Efteråt ska display också visa bilden som är färdig omvandlad med bild processen för att kunna se resultat av bildbehandlingen.

Figur 2.7: *Ett foto på Displayen PiTFT Plus, foto: Min Cheng.*

2.2.4 mDrawBot Kit

Robotarmen som ska rita ut bilden på ett papper, konstruerades utifrån ett kit skapat av företag *Makeblock* [12]. Namnet på utvecklingskitet som används är mDrawBot och den innehåller fyra olika ritrobots kit i en uppsättning. Rit roboten som valdes för detta projekt kallas *mScara*, se figur 2.8.2, då den närmast illustrera en hand passar den in till syftet. Utvecklingskitet består av Makeblock komponenter, som består av 60 komponenter, såsom motorer, balkar, hjul etc. För att styra robotarna har företag designat en programvara, *mDraw*, vilket ger användaren chansen att välja vilken bild som skall ritas ut samt val av andra alternativ för att anpassas efter behov. Se tidigare avsnitt för mer förklaring om mDraw.



Figur 2.8.2: *Ett foto på robotarmen (mdrawbot, mScare), fotograferat av Min Cheng.*

2.3 Bildbehandling

I detta projekt avser bildbehandling till att behandla en bild som en metod vilket konverterar en digitalbild från ett format till ett annat. För att kunna utföra olika operationer på bilden, som vidare antingen är för att förbättra bilden eller hämta information från den [22]. I detta projekt skall bilden anpassa till sitt ändamål vilket är att användas av robotarmen. Bildbehandlingen är en typ av signalbehandling vilket vanligtvis innebär att behandla bilden som tvådimensionella signaler och använda redan definierade signal behandlingsmetoder. I grunden är behandlingen av en bild uppdelad i tre steg:

1. Importera en bild genom optisk skanning eller digital fotografering.
2. Analysera och manipulera bilden genom att jämför data, bildförbättring eller hitta mönster som inte mänskliga ögat kan.
3. Resulterar i en omvandlad bild enligt krav alternativt en rapport om bildanalysen.

Syftet med att processa en bild kan ses på olika sätt, såsom att observera icke synliga objekt, lokalisera speciella objekt, mätta olika objekt etc. Vad än anledningen kan vara har forskningen inom bildbehandling skapat stora framsteg inom flera områden i samhället t.ex. biomedicinska bild metoder för att snabbare diagnostisera en patient, automatisera visuella inspektioner inom industrin för att hitta kritiska fel i material, satellit övervakningsmetoder för att hålla konstant koll på land och hav mm.

I detta projekt används bildbehandling eftersom att robotarmen behöver klara av att rita ut de bilder som skickas till den, lämpligast är en linjäriserad bild med få detaljer. Därför valdes en metod som kan lokaliserar de pixlarna i bilden som representerar konturer, för att systemet ska kunna dra linjer mellan dem för att få en linjäriserad bild, *edge detection*.

2.3.1 Edge Detection

Edge Detection definieras som processen att markera pixlar i en bild som ligger på de gränser där plötsliga förändringar i intensitets sker [19]. När ljuset projiceras in i en kamera för att bitvis skapa en bild, har närliggande pixlar vanligtvis samma egenskaper. Detta innebär också att pixlarna är lika i intensitet. Denna regel raderas när intilliggande pixlar projiceras från punkter som är på vardera sidan av en ocklusion gräns. Dessa pixlar kan ha olika egenskaper(färger) och resulterar i plötsliga förändringar i intensitet.

Vad som menas med intensitet inom bildbehandling är de intensitetsvärden vilket en pixel innehåller. Typiskt finns det tre stycken intensitetsvärden som representeras av de tre grundfärgerna röd, grön, blå, och varje värde ligger i ett intervall som beskriver bidraget av färgen i pixeln t.ex. röd=0, grön=0, blå=0, är en pixel som är svart färgad.

Utformandet av Edge Detection kan hänföras till de tidigaste åren av bildbehandling. I detta fall att få datorer att automatiskt kunna bearbeta och förstå all information en digital bild innehåller. Under den tiden fanns det flera personer som testade olika signal behandlingsmetoder för att hitta den främsta metoden för att lösa problemet med edge detection. Men år 1986 togs det fram en signalbehandlingsmetod som än idag bedöms som den optimalaste. Canny edge detection framtagen av John Canny, metoden som används i detta projekt för att skapa en linjäriserad bild.

2.3.2 Wolfram Mathematica

Programvara Wolfram Mathematica valdes som ett alternativ för bildbehandling. Stephen Wolfram grundad år 1987 Wolfram Research [15], en av världens mest omtyckta web mjukvara företag idag. De släppte deras första version av Mathematica år 1988 med syftet att skapa ett verktyg som alla kan använda för att förenkla beräkningar. Under år 2007, släpptes sedan wolfram alpha, en fråga-svar webbsida, som används av miljontals människor varje dag bland annat genom sina mobila applikationer.

Wolfram Mathematica är framförallt ett kommersiellt symboliskt beräkningsprogram, det används inom flera områden [16], t ex data analysering, grafiska beräkning, bildbehandling etc. Mathematica 1, deras första version av programvaran innehöll över 500 funktioner. De funktionerna finns fortfarande i dagens senaste version som nu innehåller nästan mer än 5000 inbyggda funktioner. Dessutom är Mathematica konstruerat för att kunna vara ansluten till det mesta, såsom olika databaser och anordningar.

3. Metod
Detta kapitel beskriver i tur och ordning hur arbetet har planerats för att nå det slutgiltiga resultatet samt vidare utförande av de praktiska delarna.

3.1 Projekt Preparering
Innan projektets start blev det förbestämt att arbetet skulle utföras under ca 4 månader (ca 16 veckor), därför skrevs en preliminär planering för att lyckas med att utföra projekt inom denna tidsram, se bilaga A.4. Under uppstarten bestämdes att Python skulle bli det primära språket som ska användas på en Linux plattform, eftersom Python har relativt lätt läsbarhet jämfört med andra programmeringsspråk och stödjer flera olika operationssystemet. Mer om varför Python valdes se kapitel 6.3. Efter detta steg påbörjades en kort studieperiod för att kunna använda Python vid programmering. Eftersom projektmedlemmar inte hade någon tidigare kunskap av Python eller Linux. Under inlärningsperioden ankom robotarmen och den byggdes upp. För att testades att inga problem med dess funktionalitet fanns. Inom de första 6 veckor undersöktes och studeras olika information om programmeringsspråk Python och operativsystemet Linux.

Det visade sig unders tester att programvaran för robotarmen är skriven i Python, med ett visst bibliotek tillhörande Python. För att kunna modifiera koden för programmet till robotarmen som skulle uppfylla på projektets krav, studerades detta bibliotek under inlärningsperioden. Samtidigt utfördes en undersökning av vilka bildbehandlings processer som kan ge bästa resultat för systemet.

3.2 Kodning
Efter det att tillräckligt med kunskap inhämtats, påbörjades en första modifiering av koden för programmet mDraw till robotarmen för att den automatiskt ska starta och utföra alla kommandon fram till starten av rit processen. Varje ändring som gjordes testades för att kunna se så att inga fel uppstod innan nästa modifikation skulle göras. Då lösningen till en ändring i koden inte fungerade testades olika sätt för att lösa problemet, vid okända problem eller inget lösningsförslag gick att ta fram användes tidigare liknande projekt som hjälp. Om tidigare projekt inte kunde ge giltiga svar tillfrågades personer eller grupper med närliggande kunskap om ämnet.

Efter att alla justeringar av programmet för robotarmen testats och verifierats startades programmeringen av bildbehandlingsprogrammet i en separat fil. Undersökningar av program som är öppen källkod och kunde bäst uppfylla kravet för detta projekt gjordes. Bildbehandlingsprogrammet kodades och testades konstant, en bild skickades in i programmet och den konverterade bilden analyserades. På samma sätt testades programmet till robotarmen efter varje modifikation. Olika kommandon och olika strukturer testades för att hitta den optimalaste och samtidigt undersöktes rapporter från liknanden arbeten.

Fortsättningsvis testades kameran som tillhör Raspberry Pi för att påvisa att inga hårdvarufel fanns. Därefter skrevs ett kort program till kameran för att ta en bild med specifika egenskaper.

Efter det att alla tre program testats och verifierats avseende deras funktioner, skrevs kod för få att kommunicera mellan programmen. Att samma bild som togs med hjälp av kameran fortsätter att bearbetas av bildbehandlingsprogrammet och vidare till

robotarmen. Hela systemet kontrollerades och ändringar gjordes för att anpassa de tre modulerna till varandra.

3.3 Analys

Efter att hela system testats ett flertal gånger, var nästa steg att utföra undersökningar på de delar av systemet som kan förbättras. Detta inkludera operativsystem, kamera och sättet att behandla en bild.

3.3.1 Operativsystem

För att kunna förbättra Raspberry Pi's verkningsgrad, så testades två olika operativsystem Linux och Windows. Programvaror som är Windows baserade kan ha bättre verkningsgrad än programvaror som är Linux baserade.

3.3.2 Kameraposition

För att kunna förbättra bildkvalitet, är avståndet mellan ansikte och kamera en viktig faktor som påverkar resultatet. En tabell gjordes att analysera inverkan av avståndet och positionen för kameran.

3.3.3 OpenCV/Mathematica

För bildbehandling finns det flera olika bibliotek för programmeringsspråket Python och för Linux som kan användas. Det främsta biblioteket, enligt vår bedömning är OpenCV, det alternativa programmet Mathematica. Då vissa funktioner i Mathematica fungerar i Windows miljö men ej i Linux miljö så är det nödvändigt att analysera detta mer.

4. Genomförande

I detta avsnitt presenteras resultaten som producerade under projektets genomförande. De olika programmen som kodades och de olika del resultaten de gav under olika sammanhang.

Detta är originalbilden som används för att illustrera skillnaden i olika metoder för att bildbehandling, ett foto självtagen med hjälp av systemets kameramodul:

Figur 4: *Ett foto på Erik Samaha, tagen med hjälp av systemets kameramodul*

4.1 Kamera

Efter att kamera modulen (Picamera V2.1) implemeterats för Raspberry Pi, skrevs ett program vilket tar ett fotografi med en specifik upplösning och spara undan bilden i filformatet PNG. För fullständig kod se bilaga A3.

Genom flera tester av att fotografera om och om igen med olika avstånd och kamerapositioner har det visat sig att ett avstånd mellan 20–25 cm beroende på ansiktsstorlek och med kameran riktad mot ögonen ger bästa möjliga resultat.

4.2 Bildbehandling
I detta avsnitt beskrivs de olika bildbehandlings metoder som använts.

4.2.1 OpenCV
Då OpevCV användes för att utvecka ett bildbehandlingsprogram, togs det hjälp av ett kodexempel från ett program av Adrien Rosebrock [24]. Resultat av programmet som skrevs kan ses i bilaga A.2. Nedan presenteras de viktigaste delarna:

Programmet börja först med att ladda in bilden, därefter omvandlas bilden till gråskala (färgbild i svartvitt) för att vidare applicera ett kommando GaussianBlur som tar bort högfrekventa störningar.

```
# Load the image, convert it to grayscale, and blur it slightly

image = cv2.imread('/home/pi/Desktop/mDrawBot-master/opencv/bild.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (3, 3), 0)
```

Bilden överförs sedan till en fördefinierad funktion (auto_canny). Denna funktion inleder med att hitta medianen av pixel intensiteten i bilden. Sedan med sigma, konstant bestämd av Rosebrock med simpel statistik, skall över och under gränsvärde bestämmas till funktionen *Canny*. Denna funktion är den som applicerar edge detection metoden för att linjärisera bilden.

```
def auto_canny(image, sigma=0.33):

# compute the median of the single channel pixel intensities
v = np.median(image)

# apply automatic Canny edge detection using the computed median
lower = int(max(0, (1.0 - sigma) * v))
upper = int(min(255, (1.0 + sigma) * v))
edged = cv2.Canny(image, lower, upper)

# return the edged image
return edged
```

Efter det att edge detection applicerats på bilden, sparas den undan i samma katalog som programmet för robotarmen. Resultatet från bildbehandlingen i detta program kan se ut som i figur 4.2.1. Bilden läses in från biblioteket ’potrace’, vars funktion är att omvandla bilden från filformatet BMP till filformatet SVG.

```
#Save the processed image
cv2.imwrite(os.path.join('/home/pi/Desktop/mDrawBot-
master/mDrawGui','bild.bmp'),auto)

time.sleep(1) #Delay

#Convert the processed image from BMP to SVG with potrace and save it afterwards.
os.system( "potrace -s /home/pi/Desktop/mDrawBot-master/mDrawGui/trump.bmp -o
/home/pi/Desktop/potrace-python-master/bild.svg")
```

Slutligen anropas kommandot för aktiveringen av styrprogrammet till robotarmen, *mDraw*, som överför informationen av bilden till armen för att ritas ut.

Figur 4.2.1: *Slutprodukt av bildbehandling med OpenCV.*

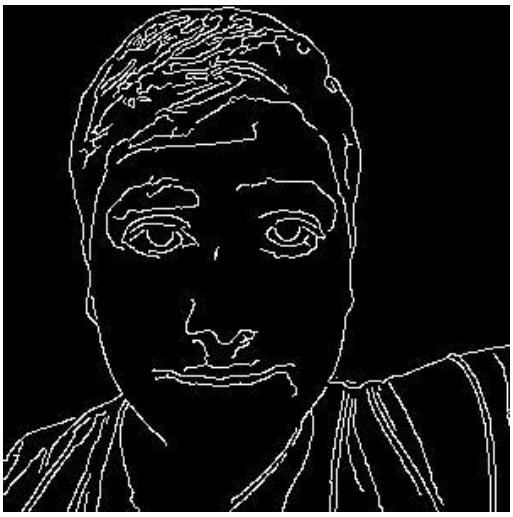
4.2.2 Wolfram Mathematica

Vid användningen av Mathematica som bildbehandlingsprogram, togs det hjälp av Wolfram Mathematica egna dokumentations center för bildbehandling [24]. Tester utfördes för att se vilka metoder som gav bästa möjliga resultat. Nedan presentera det resultat tillsammans med använd kod, den ordning de är skrivna för att ge ett visst resultat. Alla parametrar som använts är framtestade och enligt egen bedömning ger optimalt resultat.

i = originalbild

1. Hämta in orginalbilden och applicera enbart edge detection med paramterar vilket bestämmer mängden detaljer som ska framkomma.

```
//Import the picture
i = Import["Desktop/bild.png"]
//Perform edge detection
EdgeDetect[i, 2.381, 0.076]
```



Figur 4.2.2.1: Resultat från bildbehandling av originalbilden med enbart Mathematica funktion för edge detection.

2. Genom att först använda funktionen *FindFaces* för att identifiera ansikte kan man därefter utnyttja funktion *ImageTrim* för att ta bort allt från bilden förutom ansiktet och vidare applicera edge detection med samma gränsvärden:

```
//Identify face
a = FindFaces[i]
//Trim away everything but the face
a = ImageTrim[i, #] & /@ a
```

Figur 4.2.2.2: *Allt förutom ansikte borttaget*

```
//Apply edge detection
EdgeDetect [a,2.381,0.076]
```

Figur 4.2.2.3: *Resultat från att först identifiera ansiktet i bild och ta bort resterande bildelement, för att vidare utföra edge detection.*

3. Mathematica har även en funktion som kan ta bort bakgrunden på en bild. I denna test då bakgrund redan är vit finns inget att ta bort. Om man använder funktionen blir resultatet enligt nedan:

```
//Delete background
d = RemoveBackground[i]
```

Figur 4.2.2.3: *Funktionen RemoveBackground använt ger vitare bakgrund och en skugga.*

```
//Apply edge detection
EdgeDetect[d, 2.381, 0.076]
```

Figur 4.2.2.4: *Resultatet från att först ta bort bakgrunden för bilden och sedan applicera edge detection.*

4. Det finns även flera färdiga funktionen i Wolfram Mathematica som kan retuschera en bild på olika sätt. En sådan funktion som analyserades är *Sharpen*. Syftet med Sharpen är att göra bilden skarpare:

```
//Identify face
a = FindFaces[i]
//Trim away everything but the face on the image
a = ImageTrim[i, #] & /@ a
//make the image sharper with a specified pixel radius.
sharp = Sharpen[a, 10]
```

Figur 4.2.2.5: *Gör figur 4.2.2.2 skarpare.*

```
//Apply edge detection
EdgeDetect[sharp, 2.381, 0.076]
```

Figur 4.2.2.6: *Resultatet från att göra figur 4.2.2.2 skarpare och därefter applicera edge detection.*

5. Mathematica innehåller även flera olika filter som ger möjlighet att applicera ett flertal effekter på bilden. I detta projekt analyserade flera av dess filter för att se vilken effekt de kan ge då ansiktet först identifierats och vidare edge detection ska anbringas:

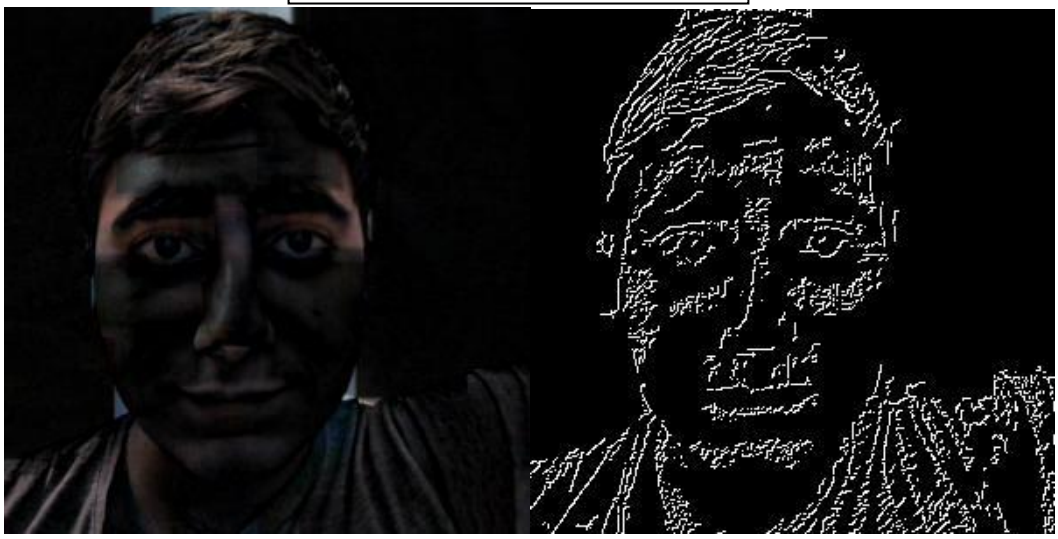
- BottomHat: Markera de mörka pixlar i bilden.

```
// Apply bottom-hat transform
bot = BottomHatTransform[image, 10]
//Apply edge detection
EdgeDetect[bot, 1.98, 0.071]
```

Figur 4.2.2.7: *Resultatet från att först använda bottom-hat transform och vidare applicera edge detection.*

- TopHat: Markerar små, ljusa pixlar från en bild

```
// Apply top-hat transform
top = TopHatTransform[i, 10]
//Apply edge detection
EdgeDetect[top, 0, 0.156]
```



Figur 4.2.2.8: *Resultatet från att först använda top-hat transform och vidare applicera edge detection.*

```
//Apply gradient filter
grad = GradientFilter[i, 2]
//Apply edge detection
EdgeDetect[grad, 2.44, 0.054]
```

Figur 4.2.2.9: *Resultatet från att först använda gradient filter och vidare applicera edge detection.*

4.3 mDrawBot

I programmet mDraw som tillhör robotarmen gjordes modifieringar till olika filer för att automatisera programmet. Följande del av en filen *mdraw.py* har modifierats:

```
def initUI(self):
        self.ui = Ui_Form()
        self.ui.setupUi(self)
        # get sys serial port and update combo box
        self.comm = "COM"
        self.serial = SerialCom.serialCom(self.commRx)
        self.refreshCom()
        self.ui.btnConnect.clicked.connect(self.connectPort)
        self.ui.btnLoadPic.clicked.connect(self.loadPic)
        self.ui.btnClearPic.clicked.connect(self.clearPic)
        self.ui.btnSend.clicked.connect(self.sendCmd)
        self.ui.btnPrintPic.clicked.connect(self.robotPrint)
        self.ui.btnStop.clicked.connect(self.robotStop)
        self.ui.btnSetRobot.clicked.connect(self.showRobotSetup)
        self.ui.portCombo.mousePressEvent = self.portComboPressed
        self.ui.labelXpos.returnPressed.connect(self.userSetPos)
        self.ui.labelYpos.returnPressed.connect(self.userSetPos)
        self.ui.lineSvgHeight.returnPressed.connect(self.userSetSvgRect)
        self.ui.lineSvgWidth.returnPressed.connect(self.userSetSvgRect)
        self.ui.lineSend.returnPressed.connect(self.sendCmd)
        self.ui.btnUpdateFirmware.clicked.connect(self.uploadFirmware)
        self.ui.btnHome.clicked.connect(self.robotGoHome)
        self.ui.btnSavePos.clicked.connect(self.savePenPos)
        self.ui.btnHelp.clicked.connect(self.linkToFAQ)

        self.connectPort()  #Connect USB-Port Automatically


        self.ui.btnHFlip.clicked.connect(self.xReflect)
        self.ui.btnVFlip.clicked.connect(self.yReflect)
        self.ui.btnRollC.clicked.connect(self.rollClockwise)
```

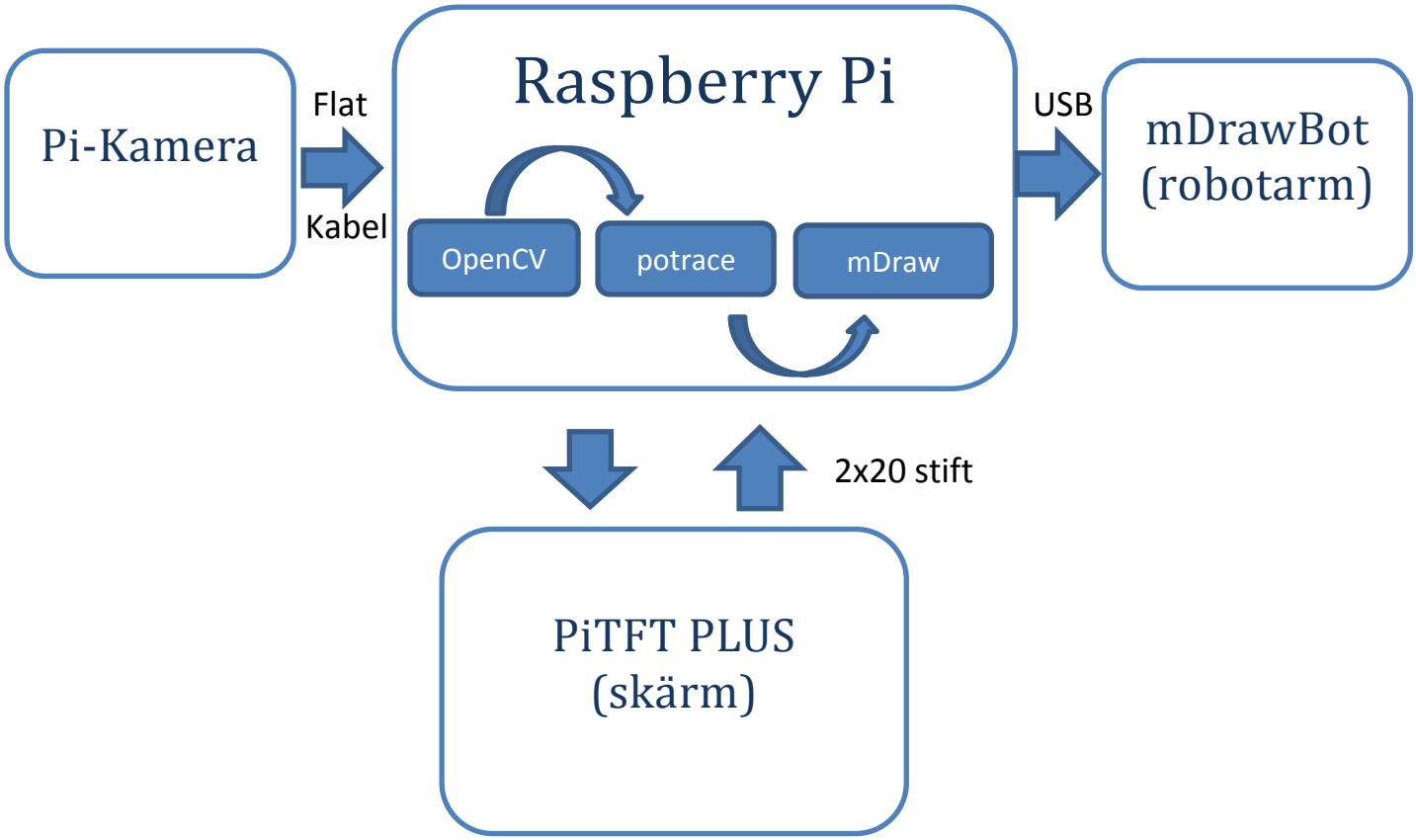
Inne i ”initUI” funktion lägger vi till en rad som är markerat med gult för att kunna ansluta till USB-port automatiskt, observera så att den inte läggs in för tidigt i funktionen, då vissa kommandon i initieringen måste vara utförda innan en vis funktion kan anropas. För att se de resterande modifikationer som gjorts i koden till robotarmen se bilaga A1.

5. Resultat

Inom ramen för detta projekt har det utvecklats ett system som kan fotografera ett ansikte och efter bildbehandling via en robotarm rita en skiss av ansiktet. I det slutliga systemet användes biblioteket OpenCv för omvandling av bilden från pixlar till linjer.
Innan funktionen edge detection används behandlingsfilter i programmet som hjälper till att producera en bättre linjäriserade bild. Gradientfilter som omvandlar bilden till gråskala (en svartvit bild). Används för att förbättra resultatet ytterligare används GaussianBlur, vilket filtrera bort högfrekventa störningar. Dessa två funktioner hjälper till att minska antalet detaljer i slutresultatet av en linjäriserade bild, men bibehåller tillräckligt av bilden ursprungliga information för acceptabel identifikation.

Operativsystemet som installerades på Raspberry Pi är Linux baserat. Robotarmen lyckades att rita ut de linjäriserade bilder som laddades till den. Skisserna som den ritade blev tillräckligt tydliga för att identifiera personen som fotades med hjälp av Pi-kameran.

Följande blockschema visar resultatet av systemet:



6. Diskussion och Analys

I detta kapitel diskuteras och analyseras nya kunskaper som lärts, upptäckta begränsningar inom systemet, samt funderingar kring de olika resultaten som producerats.

6.1 Bildbehandling

Om vi enbart fokuserar på resultatet för bildbehandlingen, ger Mathematica ett betydligt bättre slutresultat. Men då många av de funktioner som analyserats för Mathematica inte fungerar i Linux gav det i slutändan inget betydelsefullt bättre resultat än OpenCV.

Om man ska analysera Mathematica kapacitet för bildbehandling kan man se i resultatet att genom retuschera en bild först kan man förbättra ett fotografis egenskaper innan man applicera något filter enligt figur 4.2.2.5. I fall där man bara vill använda vissa delar av bilden finns det funktioner för att klippa bort resterande delar, se figur 4.2.2.2. Speciellt i vårt fall hade detta krävts för att förminska antalet detaljer som robotarmen behöver rita ut. En minskning av bilddetaljer minskar även tiden som det tar att färdigställa ritning. Mathematica innehåller flera filter som nämns och illustreras i resultatet (se figur 4.2.2.7–9). Beroende på deras effekter kan de olika filtren ge olika resultat vid tillämpning av metoden edge detection. Men hade man enbart använt gradient filter som i figur 4.2.2.9, kan man se att bilden blir näst intill optimalast för att omvandla en bild av pixlar till en linjäriserad. Enligt tidigare söker man efter en bild som är linjäriserad med så få detaljer som möjligt så att man fortfarande kan identifiera individen i porträttet.

En viktig anmärkning som gjordes vid analysen av olika metoder för att behandla en bild, är att det finns båda flera filter och funktioner för att retuschera en bild som inte gör någon skillnad i våra försök. Då vi ändå avslutar med att använda edge detection blir slutresultatet likt tidigare resultat.

6.2 Begränsningar

Som man nämnt tidigare fungerar vissa bildbehandlingsmetoder för Mathematica bara på Windows. En av de viktigaste anledning till att vissa funktioner inte fungerar på Linux, beror på Raspberry Pi och dess processorarkitektur. Arm processorarkitektur som Raspberry innehar har en Risc (Reduced Instruction Set Computing) arkitektur men Intel x86/x64 processor har en Cisc(Complex Instruction Set Computing) [23]. Dessa två processorer har helt olika arkitekturer och instruktionsuppsättningar. På grund av detta är det omöjligt att använda program och funktioner som skapats på en annan processor istället för processor man använder. Därför finns funktioner i Mathematica som bara fungerar på ett Windows system.

En väsentlig anledning till varför vi inte använde Mathematica är att när man utvecklar ett skript som anropar Mathematica, öppnas programfönster där man måste exekvera koden som utvecklats. För att fortsätta vidare behöver man stänga av Mathematica ‘s programfönster. Men eftersom displayen har upplösningen 320x 240, och programfönstret för Mathematica inte kan visa sig fullkomligt, kan man inte stänga ner Mathematica och fortsätta med hela systemet.

Projektets huvudsyfte var att utveckla ett system som kan ta en bild, konvertera bilden samt rita ut den bilden med hjälp av en robotarm, men tyvärr blev det så att kod modifiering och kodning krävde mer tid än förväntat. På grund av att programmeringsspråket Python var helt nytt för oss. När vi väl började att modifiera vissa delar programkoden till robotarmen, stötte vi ofta på något problem som vi behövde finna en lösning på. Det tog mycket tid varje gång att försöka hitta lösning från något forum, då sannolikt någon annan person har träffat på detta problem tidigare och löst detta. Om vi inte kunde hitta lösningen på detta sätt, blev alternativet att försöka analysera koden steg för steg, för att försöka hitta svaret.

6.3 Valet av Python

Innan projektet påbörjade var det ett beslut som behövdes tas om vilket programmeringsspråk som skull användas. Valet föll på Python beroende på följande:

Uppenbara enkelhet - I de andra programmeringsspråk, när man skapa en lista, behövs användningen av arrays, vektor eller pekare, men i Python, finns det en enkel funktion som kallas *list*.

Kompatibel - Utöver den kraftiga förmågan av Python:s huvudspråk och omfattande standardbibliotek, finns det även flera tredjepartsmoduler (färdiga biliotek).

Multiplattform kompatibel - Python stödjer flera olika operationssystemet, t ex Windows Mac OS X, Linux etc. Då många portabla eller mindre enheter baseras på dess operativsystem stödjer de därför Python, liksom Raspberry Pi och Mikrokontroller (en låg energi konsumerande enchipsdator)

Kod läsbarhet -Jämfört med andra programmeringsspråk såsom Java skript eller C, har Python en relativt lätt läsbarhet. Varje instruktion ha sin egen linje, det är uppenbart vilka kod rader som tillhör varandra bara genom att kolla deras indentering. Saknandet av förvirrande symbol såsom ‘{‘, ‘}’ och ‘; ‘ utspridda över koden gör det också mycket enklare att läsa igenom.

6.4 Problemlösning

En konstighet som inträffade när OpenCV installerades på Raspberry Pi, var att kompileringsprocessen fastnade utan någon anledning. Eftersom själva kompileringen skulle ta ca 2 timmar, avbröts ej kompileringsprocessen. I början förmodades att det fastnandet av processen ingick. Efter att processen fastnat ca en halv timma märkets det att hela system hade fryst sig. Vi blev tvinga att dra ut strömkabel och starta om Raspberry Pi. Efter återstart igen fortsätta initieringen från där den hade fastnat, men efter en stund fastnade den igen och det krävdes en omstart igen. Det underligaste är att om man konstant rörde musen blev det aldrig så att processen fastnade. Begrep inte om det var en tillfällighet eller inte.

En av de problemet som inte kunnat fixas är pekskärms upplösning, i vanliga fall brukar detta problemet lösa genom att installera den motsvarande drivrutinen. Men tyvärr hittades inga drivrutiner, en annan lösning som troddes skulle fungera är att ändra Raspberry Pi operativsystems output konfigurationer. Men den lösningen misslyckades också, och ett större problem inträffade på grund av detta vilket beskrivs senare i rapporten. Displayen har bara 320x240 upplösning. Anledningen till att man vill ändra upplösningen, är för att vissa program som används för detta projekt kan inte visas fullständigt på displayen. Förmodligen skulle en lite högre upplösning t ex 720p kunnat lösa detta problem.

Att omvandla bilden från PNG till SVG format var ett av de största problem som påträffades, eftersom programmet till robotarmen stödjer enbart Windows och Mac OS X. Så när man försöker implementera koden till Linux systemet, fanns en nyckelfunktion som heter *SVG-Converter* som inte fungera. Då programmet ska ladda upp en bild för att ritas ut anropas först en funktion som heter *Svg-Parser*. Funktion Svg-Parser är till för att bekräfta vilket filformat programmet kommer att läsa in. Det finns två filformat mDraw kan identifiera: SVG och BMP. Om det är i BMP går programmet vidare till en separat fil som heter SVG-Converter vilken konvertera BMP till SVG. Om omvandlingen är lyckad till filformatet SVG kommer Svg-Parser godkänna och plotta ut bilden direkt. Men Svg-Converter funktionen är väldigt komplicerat, det är bortom vår förmåga att modifiera koden då den fungera på ett Linux system. Detta blev anledningen till att vi använde biblioteket potrace för att kunna omvandla vår bild till SVG format.

6.5 Windows 10 IoT (*Internet of Things*) Core

Windows 10 IoT Core är ett operativsystem från Microsoft som syftar till inbyggda system, stödjer t.ex. en enkortsdator eller en mobiltelefon. Systemet fungerar så att programmeraren måste ansluta en externa utvecklings dator för att kunna programmera, kompilera och ladda ned programmet till datorkortet för att kunna utveckla olika applikationer.

När problemet gällande BMP till SVG omvandling uppstod, gavs ett tips från vår handledare att analysera Windows 10 IoT. Detta testas för att användas till att lösa problemet, samt att samtidigt installera och testa Mathematica. Eftersom vissa funktioner på Mathematica, som nämnts tidigare, inte fungera med Linux systemet. Men Windows 10 IoT som troddes vara som ett Windows operativsystem på en PC, visade sig ej vara det. Man kan ej installera applikationer direkt i IoT utan man måste ansluta sin Raspberry Pi till en utvecklingsdator. Men när en analysering av systemet gjordes, visade det sig att Windows 10 IoT har ganska låg kompabilitet med Pythons extensions bibliotek. T ex OpenCV har bara stöd för C# ej för Python eller för PyQt vilket mDraw är programmerat med. Eftersom det inte gick att hitta någon lösning att installera extensions bibliotek i det, så fortsättas ingen vidare utveckling med Windows 10 IoT Core.

7. Slutsats

Under perioden som detta arbete utförts har lärdomar kring Linux och Python inhämtades. Genom flera bra och bättre försök har förståelse för just vad som är optimalast, gällande operativsystem och olika programvaror, för en anordning uppbyggd med en Raspberry Pi, en Picamera v2.1 och ett mDrawBot kit. Med operativsystemet Linux, vilket fungera lämpligast med en hårdvara som Raspberry Pi och valet av Python som programmeringsspråk, blev slutsatsen att hjälpmedlet OpenCv gav den mest eftersträvansvärda bildbehandlingen. Med hjälp av dess metod Canny edge detection och inledande retuschering funktioner fick man ett önskvärt resultat. Vilket är en bild som är linjäriserade med tillräckligt många detaljer att man fortfarande kan identifiera personen i ritningen, samtidigt med att utritningstiden för robotarmen är rimlig.

Vid skedet av analys, då tester av kameran gjordes. Blev slutsatsen att ett avstånd mellan 20–25 cm (beroende på ansiktsstorlek) och med positionen av kameralinsen riktat mot en individs ögon, gav bästa möjliga resultat.

8. Förslag till fortsatt arbete och några erfarenheter

Då detta projekt har bestått av olika element, såsom uppbyggnad av en robotarm, studium av bildbehandling och programmering. Fokus under projektarbetet har legat på bildbehandlingen, vilket gör att det finns en hel del förslag till fortsatt arbete. Några nämns nedan:

- En möjlighet är att konstruera en robotarm från grunden, som ska kunna rita. Där komplexiteten av mjukvara och hårdvara för armen är anpassat till arbetes nivå.

- Man kan också gå ännu djupare i bildbehandling och skriva en egen funktion eller flera egna funktioner för att processa en bild.

Genom det här examensarbetet vi har lärt oss nya kunskaper, t.ex. på vilket sätt man hanterar en Raspberry pi, hur man installera olika operativsystem på Raspberry Pi såsom Noob eller Raspbian(olika varianter av Linux) och hur man effektiv använder mjukvaran för att skriva kod och installera program.

En viktig lärdom vi lärt oss från examensarbete är att aldrig dra ut någon komponent innan strömmen helt stängts av till en Raspberry pi, då det är hög risk att komponenter går sönder. Vår första SD-kort och Pi-kamera förstördes efter det att man dragit ut komponent direkt utan att strömmen helt stängts av. Vi blev tvungna att beställa nya komponenter och vänta en viss period in ankomst. Orsaken är fortfarande oklar för oss.

Eftersom robotarmen är ett färdigt kit, betydde detta att det ingår ett färdigt program för att driva/styra robotarmen till att kunna rita ut en bild. Men detta program som heter mDraw är ett program med grafiskt användargränssnitt, måste man använda musen för att klicka på någon ikon t ex klicka på öppna ikonen för kunna läsa bild, sedan måste man klicka annan ikon för att kunna starta utskriften av bild. Vårt mål att allting ska köras har varit automatiskt, alltså krävdes det en modifiering av koden för att kunna uppfylla projektkravet. Eftersom programmet är skrivet i Python som deras huvudprogramspråk vilket är det språk som vi fick tipsat att använda, så bestämde vi redan i början av projektet att använda Python som våra huvud skript språk. Så den erfarenhet som man kan få härifrån att det är alltid bra att fråga någon eller be om tips från någon som tidigare jobbat inom arbetsområden. Man spara massa tid istället för göra något som är tidskrävande, en ”genväg” till målet

Under detta examensarbete, har vi fått erfarenhet i att det alltid är bra att göra en säkerhetskopia på alla program, innan man fortsätter att utveckla några nya funktioner eller försöker ändra viktig konfiguration. Ett konkret misstag som vi gjorde är när vi installerade en mini pekskärm till Raspberry Pi och försökte fixa ett output problem genom att ändra operativsystemets konfiguration. Vi ändrade ett speciellt kommando som handlar om HDMI-output, efteråt när en omstarta av systemet gjordes, kom ändå ingen signal till pekskärm samt ingen bild till dataskärmen. Först försökte vi ta ut SD-kort och sätta in det till en stationär dator med Windows system och försöker ändra tillbaks kommandot. Men tyvärr kunde inte Pc:n visa allt som var sparat på SD-kortet, på grund av Linux speciella format. Vi försökte därefter använda en dator med Linux system för att komma åt den filen som ändrades och återställa den. Detta visade sig inte fungera och det bestämdes att installera SD-kortet med alla tillägg igen. Detta tog oss olyckligtvis ca två dagar att utföra.

Källförteckning

[1] Wikipedia, ”Python (programming language),” April 2017. [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[2] Wikipedia, ”General-purpose programming language,” Wikipedia.org, April 2017. [Online]. Available: https://en.wikipedia.org/wiki/General-purpose_programming_language.

[3] Wikipedia, ”Raspberry Pi,” Wikipedia.org, Februari 2017. [Online]. Available: https://sv.wikipedia.org/wiki/Raspberry_Pi.

[4] Wikipedia, ”Programmeringsparadigm,” Wikipedia.org, September 2016. [Online]. Available: https://sv.wikipedia.org/wiki/Programmeringsparadigm.

[5] Qt, ”About Qt,” wiki.qt.io, Februari 2017. [Online]. Available: http://wiki.qt.io/About_Qt.

[6] Qt, ”20 Years of Qt Code,” qt.io, 2017. [Online]. Available: https://www.qt.io/qt20/.

[7] Wikipedia, ”Grafiskt användargränssnitt,” Wikipedia.org, Augusti 2015. [Online]. Available: https://sv.wikipedia.org/wiki/Grafiskt_anv%C3%A4ndargr%C3%A4nssnitt.

[8] Wikipedia, ”OpenCV,” Wikipedia.org, Mars 2017. [Online]. Available: https://en.wikipedia.org/wiki/OpenCV.

[9] crunchbase, ”Itseez,” crunchbase.com, 2016. [Online]. Available: https://www.crunchbase.com/organization/itseez#/entity.

[10] D. Davis, ”Intel Acquires Computer Vision for IOT, Automotive,” *intel Newsroom,* MAy 2016.

[11] P. Di Justo, ”Raspberry Pi or Arduino Uno? One Simple Rule to Choose the Right Board,” *Make Magazine,* December 2015.

[12] Github, ”Makeblock-official/mDrawBot,” Github, 2016. [Online]. Available: https://github.com/Makeblock-official/mDrawBot.

[13] E. Upton, ”New 8-megapixel camera board on sale at $25,” *Raspberry Pi blog,* April 2016.

[14] P. Selinger, ”Potrace,” Privat Person, Februari 2017. [Online]. Available: potrace.sourceforge.net.

[15] Wolfram Research, ”About Wolfram Research,” Wolfram Research, 2017. [Online]. Available: http://www.wolfram.com/company/background.html?source=footer.

[16] Wolfram Research, ”Wolfram Mathematica,” Wolfram Research, 2017. [Online]. Available: https://www.wolfram.com/mathematica/.

[17] adafruit, ”PiTFT Plus 480x320 3.5" TFT+Touchscreen for Raspberry Pi,” adafruit, [Online]. Available: https://www.adafruit.com/product/2441.

[18] Wikipedia, [Online].

[19] K. Ikeuchi, ”Computer Vision A Reference Guide,” i *Computer Vision A Reference Guide*, Springer US, 2014, pp. 231-235.

[20] Wikipedia, ”Bitmap,” Wikipedia.org, Mars 2016. [Online]. Available: https://sv.wikipedia.org/wiki/Bitmap.

[21] Raspberry Pi Fondation, ”Raspberry Pi 2 Model B,” Raspberry Pi Fondation, [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[22] Engineers Garage, ”Introduction to Image Processing,” Engineers Garage, 2012. [Online]. Available: https://www.engineersgarage.com/articles/image-processing-tutorial-applications?page=3.

[23] Wikipedia, ”ARM architecture,” Wikipedia.org, April 2017. [Online]. Available: https://en.wikipedia.org/wiki/ARM_architecture.

[24] A. Rosebrock, ”Zero-parameter, automatic Canny edge detection with Python and OpenCV,” pyimagesearch, 6 April 2015. [Online]. Available: http://www.pyimagesearch.com/2015/04/06/zero-parameter-automatic-canny-edge-detection-with-python-and-opencv/.

[25] Wolfram, ” Wolfram Language & System Documentation Center,” Wolfam, 2017. [Online]. Available: http://reference.wolfram.com/language/.

Bilaga

A.1 Modifierad kod – mdraw-Python

Nedan vissa alla de olika modifikationer som gjorts i koden som tillhör robotarmen och dess program mDraw. Förändringarna är markerat i blått och borttagen kod markerat i rött:

File: mdraw.py

########## *Automatiskt ansluta USB-port med Robotarm*

```
def initUI(self):
        self.ui = Ui_Form()
        self.ui.setupUi(self)
        # get sys serial port and update combo box
        self.comm = "COM"
        self.serial = SerialCom.serialCom(self.commRx)
        self.refreshCom()

        self.ui.btnConnect.clicked.connect(self.connectPort)
        self.ui.btnLoadPic.clicked.connect(self.loadPic)
        self.ui.btnClearPic.clicked.connect(self.clearPic)
        self.ui.btnSend.clicked.connect(self.sendCmd)
        self.ui.btnPrintPic.clicked.connect(self.robotPrint)
        self.ui.btnStop.clicked.connect(self.robotStop)
        self.ui.btnSetRobot.clicked.connect(self.showRobotSetup)
        self.ui.portCombo.mousePressEvent = self.portComboPressed
        self.ui.labelXpos.returnPressed.connect(self.userSetPos)
        self.ui.labelYpos.returnPressed.connect(self.userSetPos)
        self.ui.lineSvgHeight.returnPressed.connect(self.userSetSvgRect)
        self.ui.lineSvgWidth.returnPressed.connect(self.userSetSvgRect)
        self.ui.lineSend.returnPressed.connect(self.sendCmd)
        self.ui.btnUpdateFirmware.clicked.connect(self.uploadFirmware)
        self.ui.btnHome.clicked.connect(self.robotGoHome)
        self.ui.btnSavePos.clicked.connect(self.savePenPos)
        self.ui.btnHelp.clicked.connect(self.linkToFAQ)


        self.connectPort()  #Connect USB-Port Automatically




        self.ui.btnHFlip.clicked.connect(self.xReflect)
        self.ui.btnVFlip.clicked.connect(self.yReflect)
        self.ui.btnRollC.clicked.connect(self.rollClockwise)
        self.ui.btnRollAC.clicked.connect(self.rollAntiClockwise)
##############################
```

########## *Ladda upp bilden på program fönstret och börja skicka den till robotarmen för utskrivning.*

```
self.initGraphView()
        self.robot.initRobotCanvas()
        self.setWindowTitle('mDraw')
        self.setWindowIcon(QtGui.QIcon('mDrawIcon.png'))
        self.show()
        #start refresh thread
        self.refreshThread = WorkInThread(self.sceneRefresh)
        self.refreshThread.setDaemon(True)
        self.refreshThread.start()

        time.sleep(1)     #Give some time for the USB-Port to connect.
        self.loadPic()    #Start the function for loading a picture up to the programs Window.
        self.ui.slideLaserDelay.setValue(25)
        #When a picture is uploade to the program window, begin sending it to the arm for
printing.
       self.robotPrint()
#########################
```

##########################För att ladda upp rätt bild på rätt position automatiskt, kärvs det förändringar i den definierade funktionen tillhörande detta. I den ”loadPic” funktion kan man se att vi har tagit bort en del av koden. Anledning är att original funktion stödja två filformat: SVG och BMP, men när funktioner läser en BMP filen går programmet in till showconvert funktion som koden visa nedan. Funktion är till för att omvandla BMP till SVG, men det fungerar inte på Linux systemet för processens ARM arkitekt skull. Därför bestämdes att för Linux använda en bild i SVG-format från start för att kunna skippa det här problemet.

```
def loadPic(self,filename=False):
        #if self.robot.printing:
        #   return
        #time.sleep(0.5)
        #if filename==False:

        #####Give the path to the picture
        filename ="/home/pi/Desktop/potrace-python-master//bild.svg"

        #QFileDialog.getOpenFileName(self, 'Open Svg/Bmp', '', ".svg .bmp(*.svg
*.bmp)")[0]

        #self.dbg(filename)
        #####Removed for the purpose of making the program run automatically

        #if len(filename)==0:
            #return
        #self.clearPic()

        #filetype =filename.split(".")[-1]
        #if filetype=="svg":
        #if self.vilkor==0:

######Check that the picture is in a SVG-format
        self.pic = SvgParser.SvgParser(filename,self.scene)
        self.ui.labelPic.setVisible(True)
        self.picX0 = 50
        self.picY0 = 400
        self.picWidth = 150
        self.picHeight = 150
        self.updatePic()      #####Set the position of the picture and update it on the
program window.

#####Did not work on Linux

        #elif filetype =="bmp":
        #if self.vilkor==1:
           #time.sleep(0.5)
           #self.vilkor = 0
           #self.showConverter(filename)
```

```
##############################
```
File: SerialCom.py
```
############################## När Windows används krävs förändring i denna fil
för att välja korrekt USB-port.

def serialList():
    """Lists serial ports

    :raises EnvironmentError:
        On unsupported or unknown platforms
    :returns:
        A list of available serial ports
    """
    if sys.platform.startswith('win'):
        ports = ['COM' + str(i + 6) for i in range(1)] ####Change accordingly to the right
USB-port when using Windows.

    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this is to exclude your current terminal "/dev/tty"
        ports = glob.glob('/dev/tty[A-Za-z]*')

    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')

    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
#           s = serial.Serial(port)
#           s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result
##############################
```

File: SvgConverter.py
################################ *Då programmet används på Windows fungera dess egna SVG omvandla, men med några förändringar för att fungera automatiskt.*

```
def __init__(self,uidialog,bitmapFile,sig):
        super(SvgConverter, self).__init__()
        self.bitmapFile = str(bitmapFile)
        self.ui = uidialog()
        self.ui.setupUi(self)
        self.svgout = None
        self.ui.btnConvert.clicked.connect(self.convertToSvg)
        self.ui.btnReload.clicked.connect(self.loadBitmap)
        self.ui.btnPlotToMain.clicked.connect(self.plotToMainScene)
        self.ui.slideThr.valueChanged.connect(self.thresholdChanged)
        self.thresholdChanged()
        self.setupUI()
        self.show()
        self.loadBitmap()  #### Load the bitmap automatically  to the SVG-converter window
        self.robotSig = sig
        self.convertSig.connect(self.parseConvertSig)
        self.convertToSvg() #### convert the image to SVG
        time.sleep(0.1)     #### Delay for finishing the conversion
        self.plotToMainScene()#### Plot the converted image to the program window
################################.
```

A.2 Bildbehandlingsprogram – Python
Nedanför visas bildbehandlingsprogrammet som kodades med hjälp av OpenCV.

File: opencv2.py

```
############################
# import the necessary packages
import sys
sys.path.append('/usr/local/lib/python2.7/site-packages')
import cv2
import numpy as np
from matplotlib import pyplot as plt
import os
import time
import argparse
import glob


def auto_canny(image, sigma=0.33):
        # compute the median of the single channel pixel intensities
        v = np.median(image)

        # apply automatic Canny edge detection using the computed median
        lower = int(max(0, (1.0 - sigma) * v))
        upper = int(min(255, (1.0 + sigma) * v))
        edged = cv2.Canny(image, lower, upper)

        # return the edged image
        return edged


# load the image, convert it to grayscale, and blur it slightly
image = cv2.imread('/home/pi/Desktop/mDrawBot-master/opencv/bild.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (3, 3), 0)

# apply Canny edge detection using a wide threshold, tight
# threshold, and automatically determined threshold
# wide = cv2.Canny(blurred, 10, 200)
# tight = cv2.Canny(blurred, 225, 250)
auto = auto_canny(blurred)

# show the images
#cv2.imshow("Original", image)
#cv2.imshow("Edges", np.hstack([wide, tight, auto]))
#cv2.waitKey(0)

cv2.imwrite(os.path.join('/home/pi/Desktop/mDrawBot-
master/mDrawGui','bild.bmp'),auto) #Save the processed image

time.sleep(1) #Delay
```

```
os.system( "potrace -s /home/pi/Desktop/mDrawBot-master/mDrawGui/trump.bmp -o
/home/pi/Desktop/potrace-python-master/bild.svg") #Convert the processed image from
BMP to SVG with potrace and save it afterwards.

time.sleep(1) #Delay

#Start the modified program of mDraw to make the robot arm sketch the image.
os.system('sudo python /home/pi/Desktop/mDrawBot-master/mDrawGui/mDraw.py')
############################
```

A.3 Picamera- Python
Koden för att ta bild med Pi-kameran:

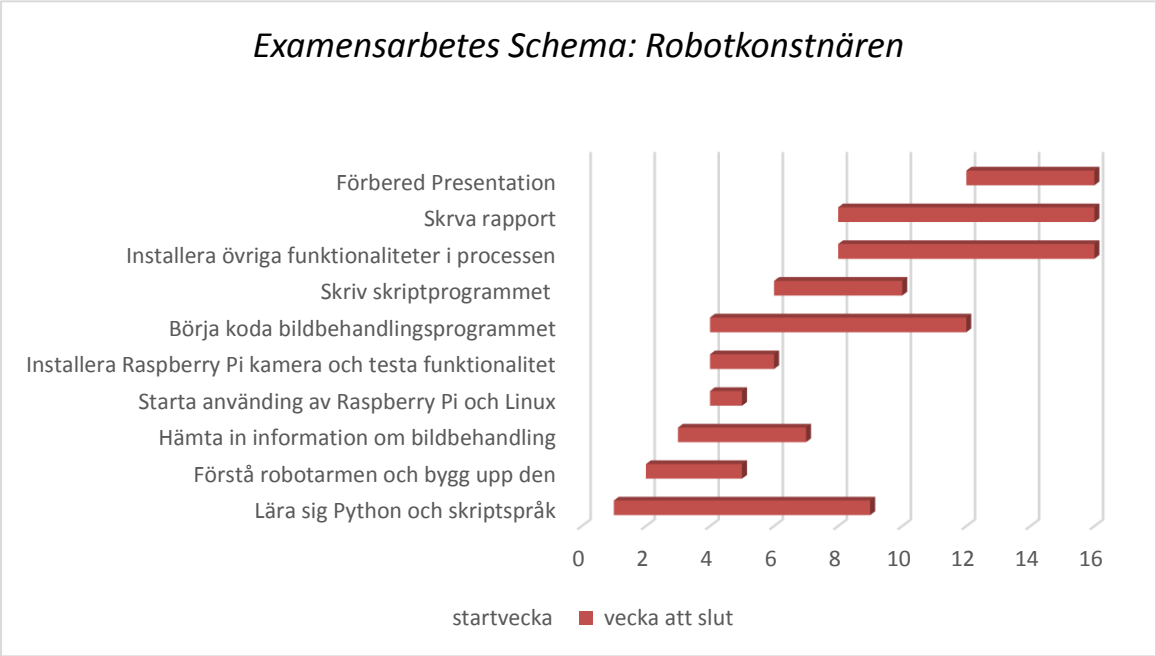
File:camera.py
```
############################
from picamera import PiCamera
import time
import os

camera= PiCamera()                    ###### give a pointer to the function PiCamera
camera.resolution = (300,300)
camera.capture('bild.png')

#execute file that named opencv2.py
os.system('sudo python /home/pi/Desktop/mDrawBot-master/opencv/opencv2.py')

############################
```

A.4 Gantt schema



Figur A.4: *En Gantt schema vilket visar ett grovt planeringsupplägg över ten planerade tidsramen.*