# QR-läsare, TNM034

Hannes Ingelhag, hanin602,
Fredrik Johnson, frejo989

19 oktober 2015

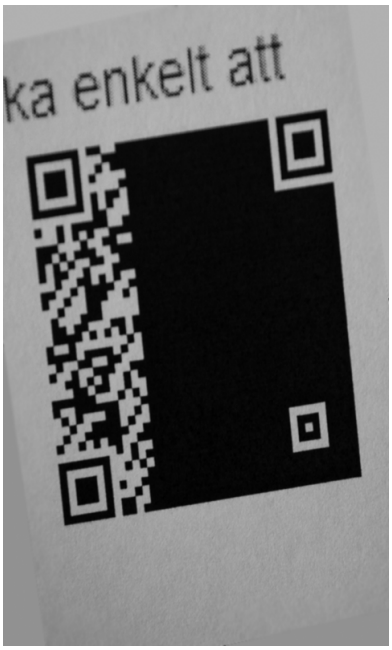
# Innehåll

# Figurer

# Kapitel 1

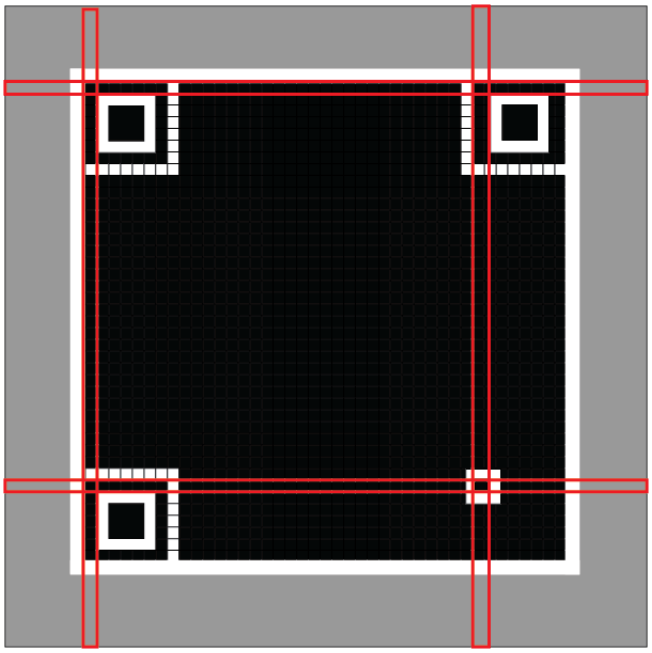
# Introduktion

Denna rapport kommer beskriva hur en Quick Response-kod-läsare kan struktureras upp så den kan läsa vilken bild som helst som innehåller en QR-kod. Det finns flera olika problem inom detta, då det är en okänd bild som skickas in i programmet kan bilden innehålla ojämna ljusförhållanden samt brus som förvärrar avläsningen. Även rotation, skjuvning och förvrängning kan göra arbetet svårare. *Figur 1.1* är ett exempel på en bild som programmet ska kunna läsa av.




Figur 1.1: *Exempelbild på vad programmet ska klara av*

Detta projekt är gjort i Mathworks programvara Matlab[1]. Det går att appliceras till de flesta programmeringsspråk, dock tillhandahåller Matlab flera inbyggda funktioner som förenklar arbetet avsevärt. QR-koden är modifierad från original versionen. Utgångs koden är av version sex med 41x41 block, dock saknar denna versionen felkorrigering samt versionsavläsning, enligt *figur 1.2*. Detta är motiverat då huvudsyftet med projektet var att finna referenspunkterna samt att extrahera informationen från QR-koden. Med denna version kommer det vara möjligt att extrahetra sammanlagt 183 tecken från den slutgiltiliga QR-koden.

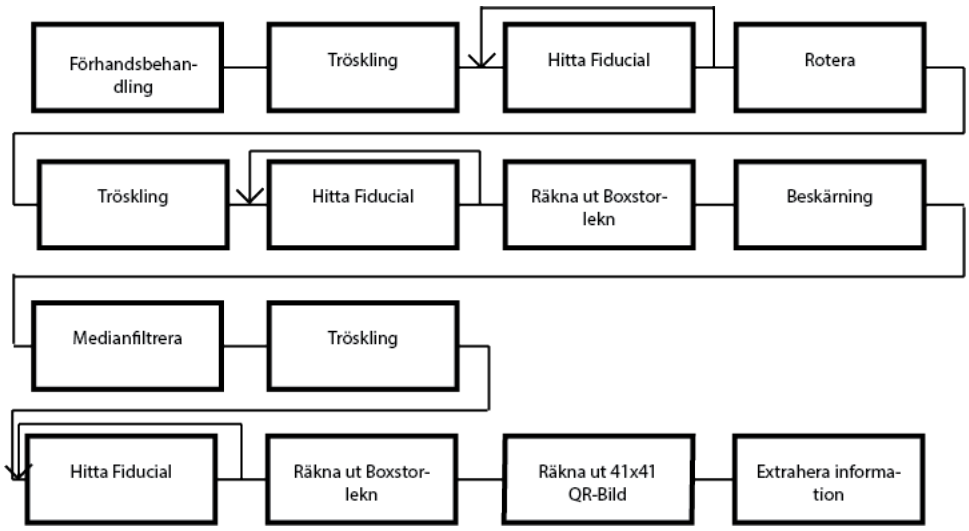


Figur 1.2: *Hur QR-koden är definerad*

# Kapitel 2

# Metod

Arbetet för att extrahera information från den okända bilden används ett specifikt händelseförlopp, enligt *figur 2.1*. De olika stegen kommer att förklaras mer i underrubrikerna till detta kapitel.



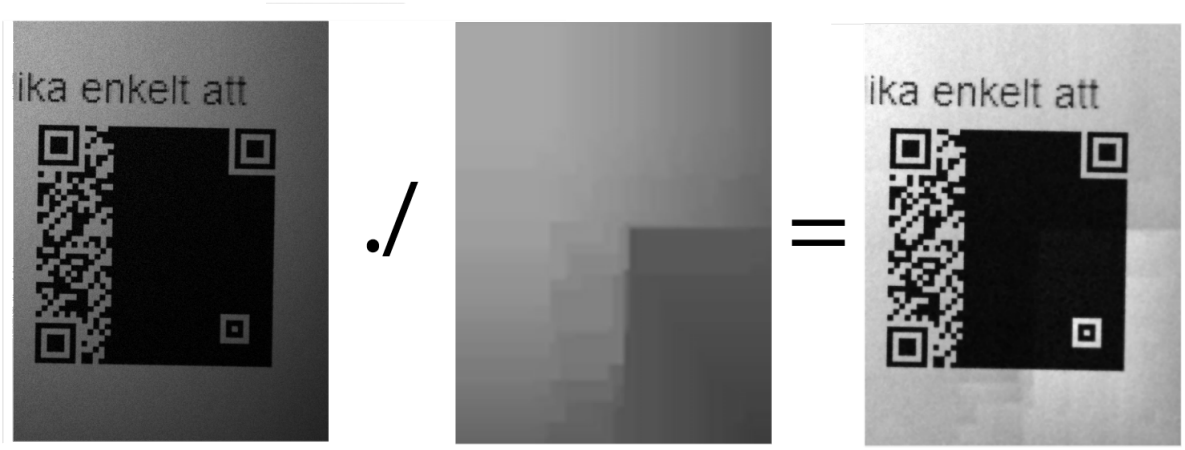

Figur 2.1: *Algoritmen för avläsningen*

## 2.1 Förhandsbehandling

Förhandsbehandlingen sker i form av filtrering av bilden. Filtreringen försöker minimera de störande element som finns i bilden och förstärka de väsentliga delarna. Det utförs två filtreringar innan något annat arbete fortskrider på bilden. Först sker en filtrering som har hand om störningar (brus, oskärpa m.m). För detta används en Matlabfunktion, *imguidedfilter*[2], denna operation minimerar brus och bevarar samtidigt skarpa övergångar (kanter) i bilden vilket är viktigt senare i processen.

Det som sedan sker har hand om att minimera ljusskillnader i bilden. Problemet med en bild som har stora ljuskillnader i sig är är att tröskla med ett globalt tröskelvärde. För att lösa detta skapas en mindre bild av vår orginalbild, detta på grund av beräkningshastigheten, då gradienten kommer se ut på samma sätt i både den större och mindre bilden. Sedan traverseras bilden med en filterkärna som vid varje iteration tar ut det lokala maxima för området som filterkärnan täcker upp. Efter operationen förstoras den nya filtrerade bilden upp till orginalstorleken.

Bilden som är trösklad kan liknas vid bildens bakgrundsljus. Varje punkt i orignalbilden divideras med respektive punkt i den filtrerade bilden. Resultatet är en bild, *figur 2.2* där ljusskillnader har blivit markant minskade.



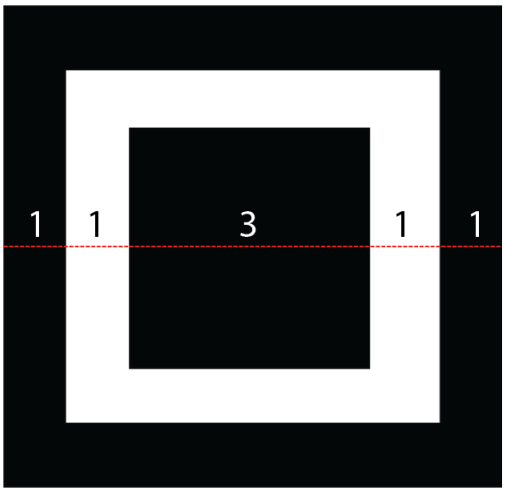

Figur 2.2: *Hantering av ojämt belyst QR-kod*

## 2.2 Tröskling

För att kommande operationer skall ge ett bättre resultat trösklas bilden och en binär bild skapas. Genom att låta tröskelvärdet vara medelvärdet mellan det minsta respektive det högsta värdet i bilden kan samma trösklingsfunktion hantera både ljusa och mörka bilder.

## 2.3 Räkna ut referenspunkter

För att finna de tre stora referenspunkterna till QR-koden används en algoritm som förlitar sig på en specifik ratio mellan olika de svarta och vita pixlarna. Ration som används är 1:1:3:1:1, *figur 2.3*, eftersom referensmönstret är kvadratiskt fungerar denna ratio även då bilderna är roterade. Processen för att hitta referenspunkterna kan delas upp i tre steg; finna linjer som passar in på ration, eliminering av dåliga linjer samt estimering av referenspunkter.




Figur 2.3: *Hur en referensfigur ser ut*

### 2.3.1 Räkna ut linjer

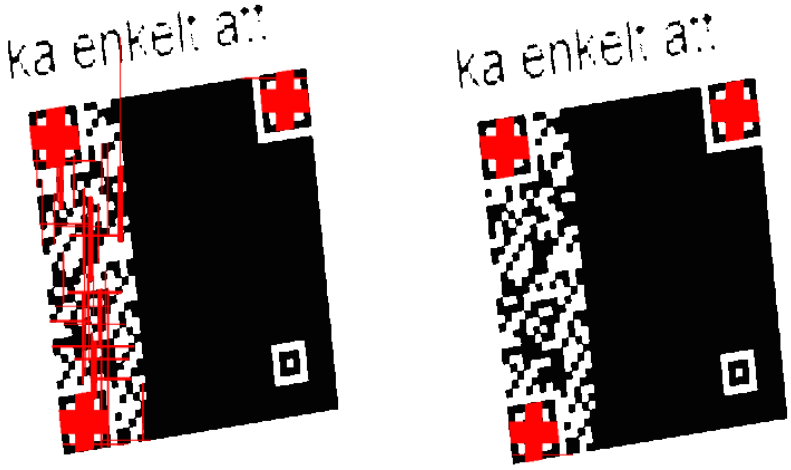
Den okända bilden itereras igenom för både x- och yled för att finna vertikala och horisontella linjer. Då bilderna är inte helt perfekta måste ett konfidensintervall användas för ration. Intervallet som användes på detta projekt var +- 0.6.

### 2.3.2 Eliminera dåliga linjer

När linjerna är framtagna är det dags att eliminera de linjer som inte beskriver en referenspunkt. Detta utförs i tre olika steg:

- **Granne som inte har någon granne**
  Om en linje inte har någon granne inom 3 pixlar så elimineras den
- **Granne som har få grannar**
  Om en grupp linjer består av mindre än ett godtyckligt antal linjer elimineras dessa
- **Linjer som korsar varandra**
  Alla linjer itereras igenom för att räkna ut hur många gånger en linje korsar en annan linje. Efter detta elimineras linjerna med få skärningar.

*Figur 2.4, a* visar hur det ser ut efter att den okända bilden har iterarats igenom för att finna linjer medan *figur 2.4, b* visar hur det ser ut efter elimineringen av dåliga linjer.

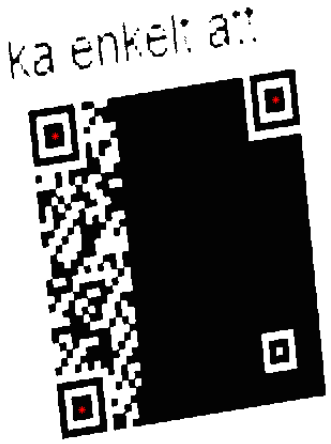


(a) *Alla linjer som hittades* (b) *Efter Elimination*

Figur 2.4: *Referensmönster*

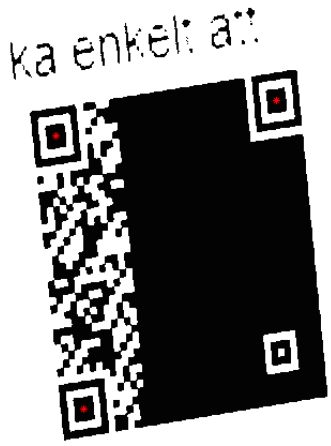
### 2.3.3 Estimering av referenspunkter

Efter eliminationen delas bilden upp i tre delbilder; Nordöstra, Nordvästra, Sydvästra. Detta för att finna de tre referenspunkterna lättare. Ett medelvärde av x-värdet på de horisontella linjerna samt ett medelvärde av y-värdet för de vertikala linjerna räknas ut för respektive delbild. Dessa punkter används för att finna mittpunkten för varje referensfigur. *Figur 2.5* visar hur punkterna ser ut efter estimeringen.



Figur 2.5: *Mittpunkter efter estimering*

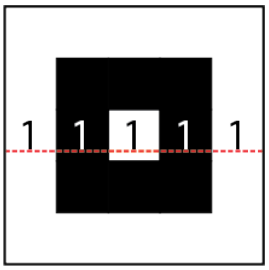
Som figuren visar ligger inte dessa punkter exakt i mitten av figurerna. Detta löses genom att använda den inbyggda Matlabfunktionen *centroids*[3], vilket räknar ut mittpunkter i objekt som existerar i bilden. Avståndet mellan punkterna, som funktionen returerar, och avståndet till de estimerade punkterna räknas ut. Den centerpunkten som har lägst avstånd blir de nya referenspunkterna enligt *figur 2.6*.



Figur 2.6: *Slutgiltliga referenspunkter*

### 2.3.4 Finna justeringsfiguren

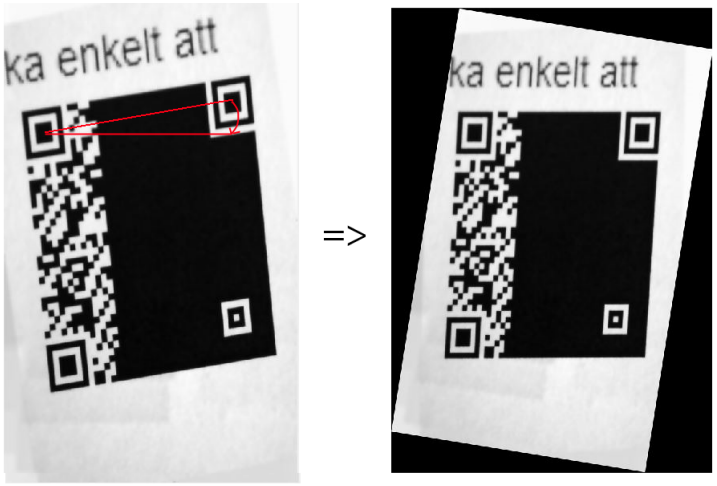
Tillvägagångssättet för att finna justeringsfiguren är väldigt likt det sätt referenspuntkerna hittas, det som ändras är det mönster som eftersöks. Ration som används är nu 1:1:1:1:1 som visas i *figur 2.7*. Likt referenspunkterna fungerar denna ratio även om bilden skulle vara roterad eller ej. Konfidensintervallet som användes är +- 0.5.




Figur 2.7: *Justeringfigur*

## 2.4 Rotering

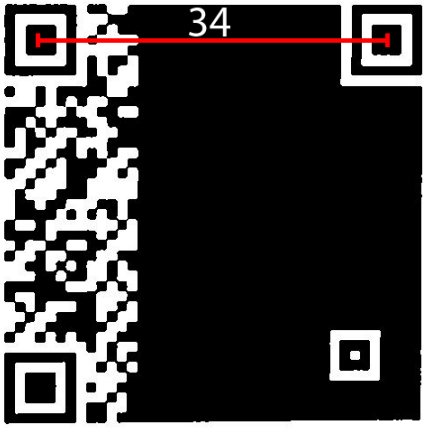
Det appliceras en rotering på alla bilder, främst för att transformation skall bli bättre, då det blir ett mindre element att ta in i transformationsmatrisen. Vilket i sin tur leder till en bättre avläsning av QR-koden. De två övre referenspunkterna lokaliseras och roterar upp den högra referensfiguren så de två ligger horisontellt med varandra, *figur 2.8*.




Figur 2.8: *Rotation*

## 2.5 Räkna ut boxstorlek

Boxstorleken definerar hur stor varje kvadratisk del i QR-koden är. För att bestämma storleken på boxen beräknas avståndet mellan två refernspunkter, enligt *figur 2.9*, som sedan divideras med en perfekt QR-bilds avstånd, vilket är 34.




Figur 2.9: *Avståndet mellan två referenspunkter*

## 2.6 Beskärning

För att utföra en mer korrekt transformering utförs en beskärning med hjälp av de intressanta punkterna i bilden, nämligen referenspunkterna. Bilden beskärs med 10 boxenheter åt alla riktningar från referenspunkterna. *Figur 2.10, a* visar en bild före och *figur 2.10, b* efter beskärning.

(a) *Före beskärning* (b) *Efter beskärning*

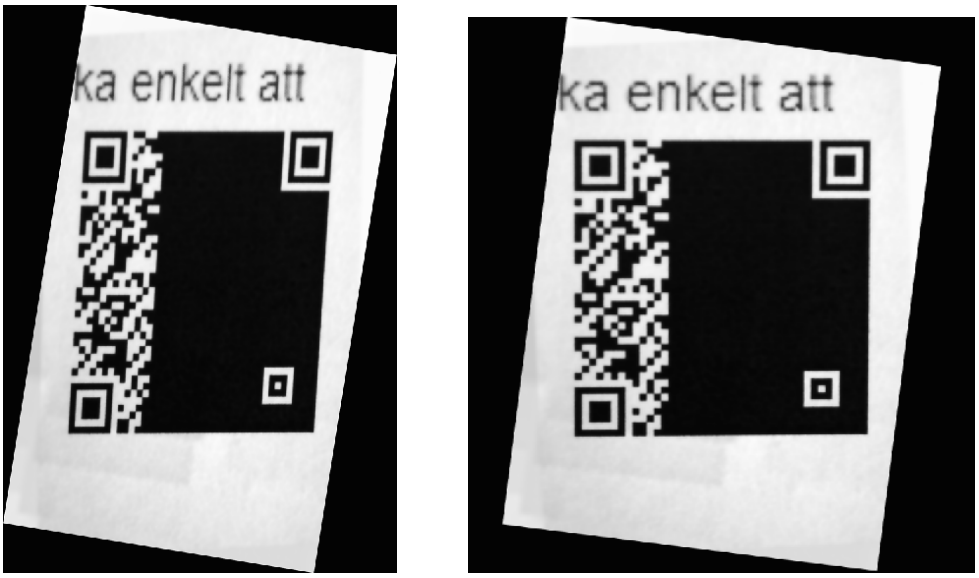
Figur 2.10: *Beskärning*

## 2.7 Transformering

På grund av geometriska förvrängningar måste bilden transformeras för att informationen från QR-koden ska kunna extraheras. För att göra en transformation används referenspunkterna samt justeringspunkten som är uträknade i orignalbilden. Dessa ska transformeras till nya referenspunkter i ett nytt koordinatsystem. Dessa nya referenspunkter tas ut på följande sätt:

1. Utgå från den nordvästra referenspunkten och sätt ut den i det nya koordinatsystemet.
2. Räkna ut avståndet från nordvästra till nordöstra-referenspunkten samt nordvästra- till sydvästra-referenspunkten och ta det minsta avståndet av dessa.
3. Använd detta avstånd och sätt ut referenspunkter i det nya koordinatsystemet för de nordöstra och sydvästra punkterna, vinkelräta mot den nordvästra punkten.
4. Den sista punkten i det nya koordinatsystemet, justeringspunkten, räknas ut genom att använda den nya punkten för nordöstras x-värde minus 3 box-storlekar samt den nya punkten för sydvästras y-värde minus 3 box-storlekar.

När punkterna i det nya koordinatsystemet är uträknade används Matlabfunktionen *fitgeotrans*[4] för att räkna ut transformationsmatrisen samt funktionen *imwarp*[5] för att göra transformationen. Skillnaden mellan orginalbilden samt den transformerade bilden går att studera enligt *figur 2.11, a* och *figur 2.11, b*.




(a) *Före transform* (b) *Efter transform*

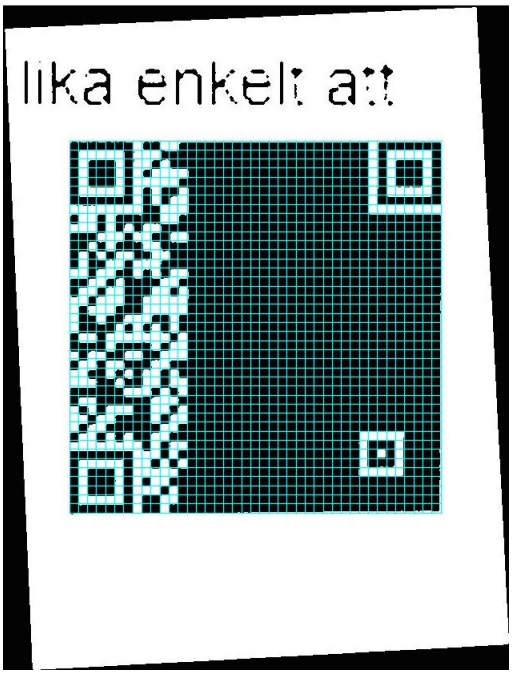
Figur 2.11: *Transform*

## 2.8 Medianfiltrering

Efter att transformations-operation har applicerats på bilden uppstår det artefakter i form av diagonala linjer. Dessa linjer kan skapa ett problem då QR-koden skall avläsas, genom att uttnytja en till filtrering i form av *medfilt2*[6] försvinner även dessa och bilden behåller fortfarande god kvalité

## 2.9 Skapa en perfekt QR-Kod

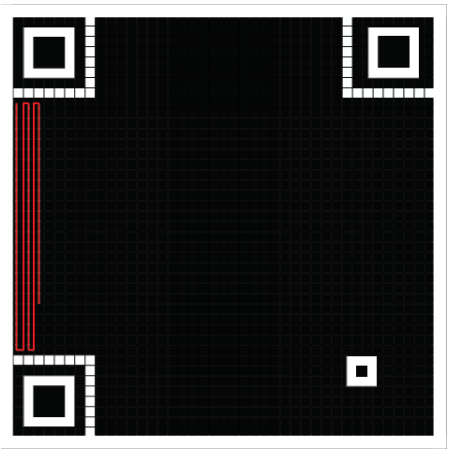
Då boxstorleken är känd sen tidigare kan nu en ”perfekt” QR-kod skapas, det vill säga en bild som är 41x41 pixlar stor och varje pixel är antingen vit eller svart. Detta sker genom att placera startposition i högra hörnet på QR-koden och sedan måla upp ett 41x41 rutnät över hela QR-bilden, *figur 2.12*. Då detta rutnätet är definerat itereras rutnätet igenom där varje medelvärde tas ut i den boxen. Beroende på detta medelvärde skapas det ett värde för motsvarande pixel som antingen är svart eller vitt i den perfekta bilden.




Figur 2.12: *Grid för QR-kod*

## 2.10 Extrahtera informationen

När den perfekta QR-koden är framställd är det dags för det sista steget, att extrahera informationen. Detta utförs genom att läsa QR-koden uppifrån och ner, *figur 2.13*, och beroende på färg placeras en 1 eller 0 in i en matris med 183rader(antalet tecken) och 8 kolumner(antal bitar). För att räkna ut vilket ASCII-värde det blir används en inbyggd funktion i Matlab men går att räkna ut enligt *figur 2.14*. Där den undre raden är 8 bitar för att bilda en bokstav, varje bit multipliceras med motsvarande tal i övre raden för att sedan summeras ihop till ett decimaltal.



Figur 2.13: *Såhär läses QR-koden av*

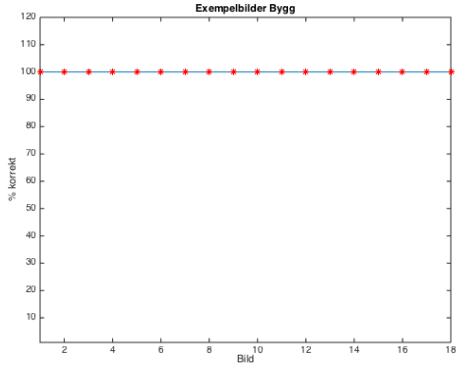
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

Figur 2.14: *Konvertering mellan binärt och decimaltal*

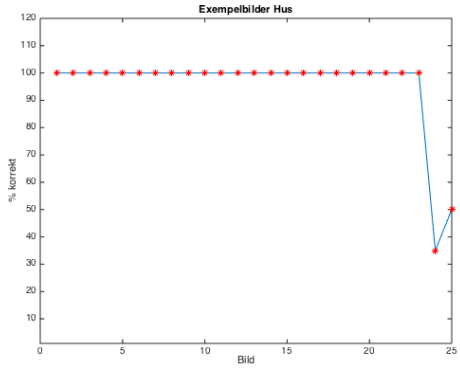
# Kapitel 3

# Resultat

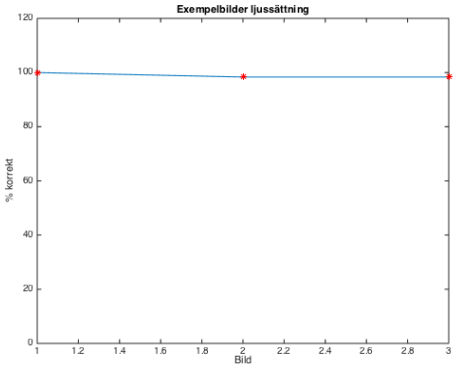
Resultatet som uppnåtts på de testbilder som varit tillgängliga går att se i *figur 3.1*. Förutom två bilder i *b* är resultatet bra, 100% korrekt.




(a) *Resultat Bygg*




(b) *Resultat Hus*




(c) *Resultat Ljussättning*

Figur 3.1: *Resultat*

# Kapitel 4

# Diskussion

## 4.1 Reflektioner över arbetet

Som resultatet visar kan vår QR-läsare läsa av bilder med problem som starkt roterade, ojämn ljussättning samt bilder som är transformaterade. Dock som det stod i resultatet blir två av bilderna inte alls lika bra som de andra. Dessa två bilder är stora bilder innehållande en liten QR-kod. Vi hade större problem med denna innan vi beskärde bilden, men även efter denna operation blir inte resultatet helt korrekt.

Från början använde vi oss av ett eget sätt att räkna ut transformationsmatrisen samt utföra själva transformationen som har implementerats i en föregående kurs på Linköpings Universitet, TNM087 - Bildbehandling och bildanalys. Dessa ersattes senare i utvecklingsprocessen av de två inbyggda Matlab funktioner som utförde dessa två steg som gjorde att bilderna förbättrades avsevärt efter operationen.

Svåraste delen i utvecklingsarbetet var att finna en tillräckligt bra stortering av de linjer som passar in på ration(1:1:3:1:1 för referensfigurerna samt 1:1:1:1:1 för justeringspunkten). Det är fortfarande inte perfekt och beror på konstanta variabler som har försökts att kalibereras för att passa alla bilder. Detta hade kunnat gjorts på ett mycket bättre sätt.

## 4.2 Förbättringar av programvaran

För att skapa en bättre transformation skulle det behövas fler punkter från orignalbilden. Detta skulle kunna exempelvis vara hörnpunkterna. Tyvärr har vi inte kunnat finna dessa på ett sätt som ger en exakt position för olika bilder och därav fått en transformationsmatris som endast förlitar sig på fyra punkter och inte är perfekt.

För att förbättra resultatet skulle även en funktion som kollar på de referenspunkter som räknas ut är rimliga värdet. Fall de ligger helt utanför ramarna skulle det kunna ändras några parametrar och sökningen skulle ske igen. Hur dessa paramentrar skulle kunna utformas är i dagsläget oklart.

I den nuvarande programvaran används en relativt enkelt gobal tröskling, nämligen summan av max- och minvärdet dividerat med två. Det skulle kunna förbättras till exempelvis Otsu's eller liknande algoritm.

Den boxstorleken som tas ut beror för tillfället på det minsta avståndet mellan två referenspunkter. Vi litar alltså på att vår transformation kommer göra QR-koden till en kvadrat, samma längd i x-led som i y-led. För att förbättra detta och ge en mer korrekt läsning borde vi använda en mer anpassad storlek i respektive led.

För tillfället transformeras bilden endast en gång. Det skulle möjligvis kunna bli en mer exakt resultat i slutbilden om bilden går igenom flera transformationer tills referenspunkterna ligger helt perfekt och i linje med varandra.

# Referenser


[1] Matlab, 2014-12-09
*http://se.mathworks.com/*

[2] Imguidedfilter, Guided filtering of images - 2014-12-09
*http://se.mathworks.com/help/images/ref/imguidedfilter.html*

[3] Centroids, Specifies the center of mass of a region - 2014-12-09
*http://se.mathworks.com/help/images/ref/regionprops.html#bqkf8ln*

[4] Fitgeotrans, Fit Geometric transformation to control point pairs - 2014-12-09
*http://se.mathworks.com/help/images/ref/fitgeotrans.html*

[5] Imwarp, Apply geometric transformation to image - 2014-12-09
*http://se.mathworks.com/help/images/ref/imwarp.html?searchHighlight=imwarp*

[6] Medfilt2, 2-D median filtering - 2014-12-09
*http://se.mathworks.com/help/images/ref/medfilt2.html*