Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Automatisering og signalbehandling	Vårsemesteret, 2017 Åpen
Forfatter: Jens Førsund	 (signatur forfatter)
Fagansvarlig: Karl Skretting Veiledere: Morten Mossige, Ståle Freyer	
Tittel på masteroppgaven: Estimere posisjon ved hjelp av QR-koder Engelsk tittel: Estimating position using QR-codes	
Studiepoeng: 30	
Emneord: QR-kode Kamera Posisjonsestimat ZBar	Sidetall: 88 + vedlegg/annet: 1 Stavanger, 15. Juni / 2017

Universitetet
i Stavanger

Estimere posisjon ved hjelp av QR-koder

Masteroppgave våren 2017

av Jens Førsund

Forord

Det rettes en takk til ABB på Bryne for inspirasjon til oppgaven, en takk til Universitetet i Stavanger for lån av utstyr og laboratorium. Veiledere Morten Mossige, Ståle Freyer og Karl Skretting takkes for støtte gjennom prosjektet.

Sammendrag

Oppgaven som er beskrevet i denne rapporten går ut på å lage et system som skal finne senter av et bilde oppgitt i et eksternt koordinatsystem i rommet. For å løse dette trengs det punkter som er kjent både i romkoordinatsystemet og i bildets koordinatsystem. Det er brukt QR-koder i motivet, og disse QR-kodene inneholder informasjon om sin egen plassering i romkoordinatsystemet samt sin fysiske størrelse. Disse QR-kodene kan leses fra bildet som er tatt med et kamera. Kameraet er koplet til en PC slik at punktene både i romkoordinater og bildekoordinater blir kjent for programvaren som kjører algoritmene.

Systemet som er laget er et resultat av flere algoritmer som er testet mot hverandre. Det er laget og testet algoritmer for på forhånd å estimere nødvendige parametere til posisjonsestimatet. Deretter er ulike algoritmer for å estimere posisjonen til senter av bildet laget og testet. En algoritme som baserer seg på vinkler i bildet har utmerket seg som generelt bedre og mer robust enn de andre som er testet. En annen algoritme som baserer seg på både vinkler og avstander foretrekkes om det er mange punkter tilgjengelig for estimering.

Etter de beste algoritmene er valgt ut har systemet blitt testet i forhold til påvirkning fra linseforvrengning, ulike størrelser på QR-kodene, ulike objektiv på kameraet, ved reduksjon av oppløsningen og ved ulike vinkler på kameraet. Systemet er testet både statisk og i bevegelse over en matrise av QR-koder for å finne presisjonen. For at systemet lett skal kunne adapteres til ulike avstander mellom motiv og kamera, har alle parametere som er relative til testoppsettet også blitt regnet om til en funksjon av avstanden.

Hovedmålet er å gi en grunngitt anbefaling av hvilken algoritme som fungerer optimalt til å estimere bildet sitt senter i romkoordinater. Dette er gjort og det er satt opp et systemoppsett som er testet og skal fungere direkte. Systemet som er utviklet og testet i denne rapporten er således ferdigstilt, og det kan settes direkte i en applikasjon der et slikt posisjonsestimat er nødvendig. Brukeren vil umiddelbart vite hvilken presisjon som kan forventes.

Innhold

Figurliste

Tabelliste

1. Introduksjon

1.1 Bakgrunn for oppgaven

Denne oppgaven er utarbeidet i samarbeid med ABB på Bryne. Utgangspunktet til oppgaven er et ønske om å kunne verifisere banen en industrirobot beveger seg i. Målet med et slikt system er å verifisere at den posisjonen roboten får kommando om å være i, eller bevege seg til stemmer med det roboten virkelig gjør. Det er da nødvendig å få en måling av roboten sin posisjon uavhengig av å lese robotens egne parametere.

ABB bruker i dag et manuelt system der en penn blir brukt som verktøy på roboten. Den tegner en strek på et ark når roboten beveger seg, og streken kan i etterkant måles og kontrollers. For å effektivisere denne verifiseringsprosessen, ønsker ABB å lage et system som kan gjøre dette digitalt, og det blir foreslått å bruke et kamera som verktøy istedenfor penn.

Problemet som skal løses er å finne ut hvor kameraet ser, eller mer presist hvor sentrum av synsfeltet til kameraet er. Fremgangsmåten som følges i prosjektet er å finne noe i synsfeltet som har en kjent posisjon og bruke det som referanse i forhold til sentrum av synsfeltet. I dette prosjektet er det valgt å bruke QR-koder som slike referansepunkter. Fordelen med å bruke QR-koder er at de er konstruert slik at de har en datastreng som er lesbar for programvare på en PC, mobiltelefon eller lignende. QR-kodene som brukes i prosjektet har datastrenger som inneholder QR-koden sin egen posisjon i romkoordinatsystemet. Ved å lese datastrengen i tillegg til å finne QR-koden fysisk i bildet, finnes det et referansepunkt i bildet som senterposisjonen kan refereres til. I prosjektet blir det brukt flere QR-koder, og de er plassert i en matriseform. Et eksempel på et utsnitt av en slik matrise finnes i Figur 1.

En av de interessante poengene med å løse dette problemet med QR-koder, er at QR-koden sin posisjon kan defineres akkurat slik som ønsket. I dette prosjektet er koordinatsystemet begrenset til en liten overflate med origo i hjørnet av matrisen. Det er ikke videre komplisert å definere posisjonen relativ til et helt rom. Om ønskelig kan QR-kodens posisjon defineres i forhold til lengde- og breddegradene til jordkloden, og da blir synsfeltet referert til eksakt hvor i verden kameraet ser.

Bruksområder

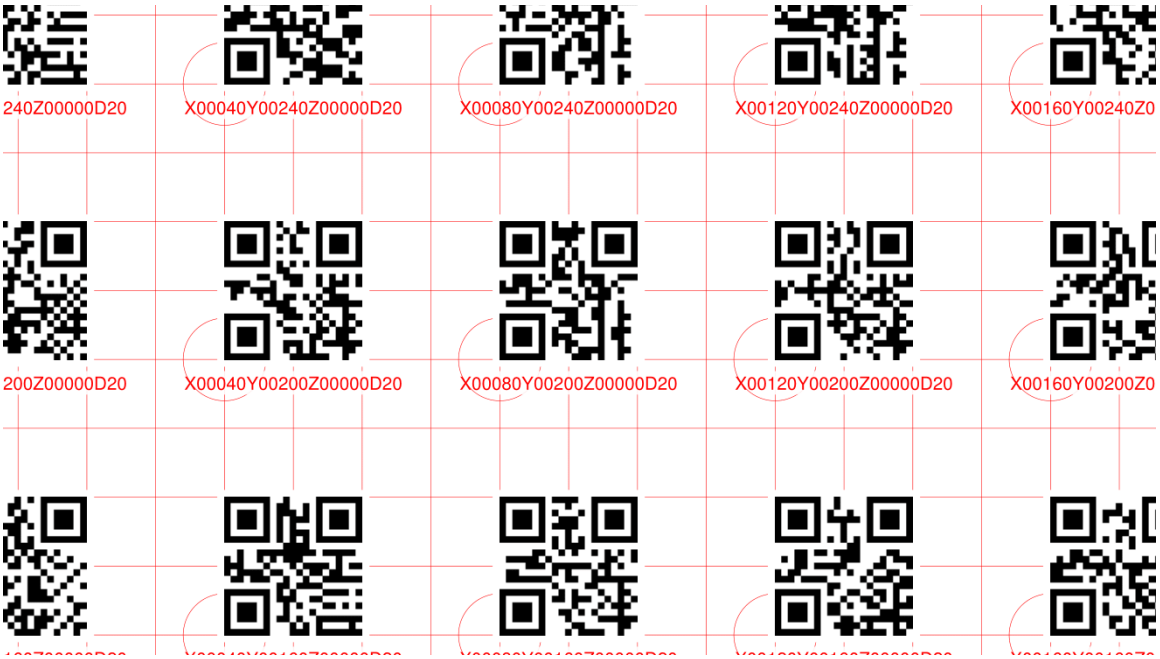
Inspirasjonen til prosjektet kommer som nevnt fra robot, men det systemet som er beskrevet i denne rapporten er ikke begrenset til bruk på robot. Det kan sees i sammenheng med de fleste applikasjoner hvor formålet er å logge posisjonen eller bevegelsen til et objekt ved hjelp av et kamera.

Systemet kan for eksempel brukes som del av selvkalibrerende systemer hvor en robot selv kan sjekke om dens egen posisjon er riktig og eventuelt kalibrere seg selv ved avvik.

Et annet formål er automatisk synkronisering av flere bevegelige maskiner eller roboter som skal jobbe sammen om oppgaver. Dette kan for eksempel gjøres ved at den ene holder kamera og den andre holder en QR-matrise.

Ved navigering for roboter innendørs hvor GPS ikke er tilgjengelig, kan QR-koder brukes som referansepunkter rundt i rommet.

Systemet kan også sees i sammenheng med Argumented Reallity (AR), hvor det er behov for presise referanser i det virkelige rommet. AR er kommet mer og mer ut i markedet de siste året, og det er naturlig at det kommer applikasjoner som krever enda høyere presisjon enn hva som finnes i dag.



Figur 1. Dette er et utsnitt av en QR-matrise som er brukt i dette prosjektet. Datastrengen i QR-koden inneholder dens egen posisjon og størrelse oppgitt i millimeter. Datastrengen er også printet i tekst for visuell kontroll. Eksempel: QR-koden i midten inneholder følgende data: x = 80 mm fra origo, y=200 mm fra origo og dimensjonen er 20x20 mm. Origo av QR-matrisen er nede til venstre og er ikke synlig i dette utsnittet.

1.2 Oppgavetekst

Oppgaven går ut på å lage et system som skal finne senter av et bilde oppgitt i et koordinatsystem i rommet. Motivet i bildet skal inneholde en overflate hvor det er festet QR-koder som inneholder informasjon om sin egen posisjon i forhold til dette koordinatsystemet. De skal også inneholde informasjon om sin egen fysiske størrelse.

Systemet som skal lages skal være et resultat av flere metoder som testes og måles opp mot hverandre. Hovedmålet er å kunne gi en grunngitt anbefaling av hvilken metode som fungerer optimalt, eller best til formålet.

Betingelse

Det settes i utgangspunktet en betingelse for systemet, og den er at kameraet hele tiden skal stå normalt mot QR-matrisen. Det er likevel ønskelig å se på hva resultatet av å utfordre denne betingelsen er.

1.3 Relatert arbeid

Perspective-n Point Problem

Perspective-n Point Problemet (PnP) er et område som har vært forsket på helt siden 1980-tallet. Problemet går ut på å estimere et kamera sin posisjon i rommet basert på punkter i bildet. Disse punktene har kjent plassering i bildet, og de har kjent plassering i rommet [1].

Dette problemet er nært beslektet med problemet i denne oppgaven. PnP har som mål å estimere kameraets posisjon i rommet. Denne oppgaven ønsker å estimere bildets senter i rommet. Blir dette sett på fra kameraets koordinatsystem, så er forskjellen mellom de to problemene en offset i z-retning som tilsvarer avstanden mellom kamera og motiv. Estimater av x- og y-koordinatene blir de samme. En løsning av problemet, EPnP, fra nyere tid er undersøkt nærmere i kap. 2.7.

Tidligere oppgave ved UIS

Det er gjort et lignende arbeid ved Universitetet i Stavanger tidligere [2]. Rapporten har noe uklare resultater og bruker ikke helt den samme fremgangsmåten som dette prosjektet. Det er likevel likheter ved prinsippet om å bruke en QR-matrise med innebygde romkoordinater. Det samme biblioteket for å lese QR-koder, ZBar, er også brukt.

Noen interessante resultater kan hentes fra dette prosjektet. Det er gjort en test med fokus på oppløsning. Her ble det konkludert med at å redusere oppløsningen til mellom ½ og ¼ gav bedre resultat med tanke på antall detekterte QR-koder. Det var benyttet et kamera med oppløsning på 1936 x 2592 piksler. Oppsettet gir ingen informasjon om avstanden fra kamera til bildet. Det ser ut til å være testet med litt tilfeldige avstander og vinkler.

Resultatene viser også at antall QR-koder i bildet har en påvirkning. Med 6 QR-koder i bildet, detekteres alle. Med 20 og 40 QR-koder i bildet, oppdages ca. halvparten. Dette kan også ha en sammenheng med at testene er gjort slik at QR-matrisen dekker hele bildet. Desto flere QR-koder, desto mindre størrelse har de i bildet.

Disse resultatene gir en indikasjon på hva som kan forventes, men videre i dette prosjektet blir også disse parameterne testet på nytt.

Relatert bacheloroppgave ved Universitetet i Stavanger

Våren 2017 har det pågått en bacheloroppgave parallelt med denne masteroppgaven på Universitetet i Stavanger [3]. Den er svært relatert da den handler om synkronisering av to bevegelige deler i et robotisert lakkeringssystem. Der er det også brukt QR-kode og kamera for å estimere posisjonene relativt til hverandre. Bacheloroppgaven er mye mer praktisk rettet og kan sees på som en relevant og reell applikasjon til systemet som er utviklet i denne masteroppgaven. Masteroppgaven går dypere i å optimalisere posisjonsestimatet for å kunne lage et mest mulig presist system som kan brukes til slike applikasjoner.

1.4 Rapportinndeling

Rapporten dokumenterer arbeidet som er gjort i prosjektet og er delt inn i følgende hovedkapitler:

Teoridel

Denne delen tar for seg teoretisk bakgrunn for algoritmene som er utarbeidet, og annen teori som er nyttig å kjenne til i sammenheng med oppgaven. Her er også informasjon rundt eksterne biblioteker som er tatt i bruk. Om leseren er godt kjent med kamerateori, QR-koder, trilaterasjon og triangulering kan dette kapittelet med fordel hoppes over og heller brukes som et oppslagsverk.

Algoritmer som er utviklet i dette prosjektet

Kapittelet inneholder algoritmer som er utviklet for å fastsette rotasjonen romkoordinatsystemet har rundt z-aksen til kameraet, og algoritmer som er utviklet for å fastsette hvor stor en piksel er i romkoordinatsystemet. Til slutt kommer algoritmer som er utviklet for å estimere posisjonen til senter av bildet i romkoordinatsystemet.

Forsøk og resultater

Dette kapittelet viser tester og resultater som er utført. Testene er gjort av algoritmene som er beskrevet i det foregående kapittelet. Det er flere tester av ytre påvirkninger på systemet, og til slutt kommer tester som viser optimal presisjon i systemet både statisk og i bevegelse.

Diskusjon

Dette kapitlet vurderer resultatene fra testene og anbefaler på grunnlag av det hvordan et systemoppsett bør være. Videre arbeid blir diskutert og til slutt kommer en konklusjon.

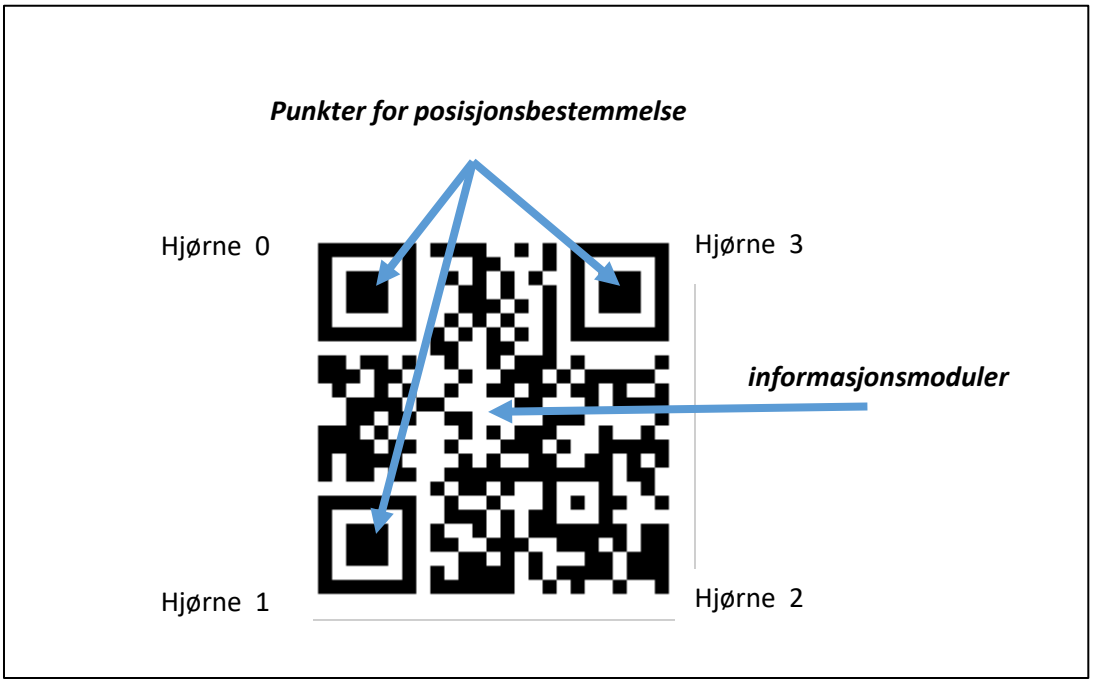
2. Teoridel

Teoridelen av rapporten inneholder teoretisk bakgrunn for algoritmene som er utarbeidet, og annen teori som er nyttig å kjenne til i sammenheng med oppgaven. Her er også informasjon rundt eksterne biblioteker som er tatt i bruk.

2.1 QR-kode og QR-kodeleser

QR-kode

Quick Response code, eller QR-kode, er en todimensjonal strekkode som har opphav fra Japan. QR-koden ble utviklet av Denso Wave for bruk i dagligvareindustrien på 1990 tallet. Til forskjell fra standard strekkode som bare hadde kapasitet på 20 tegn, var QR-koden utviklet for å kunne inneholde både alfanumeriske og japanske tegn. Mye av skylden for populariteten til QR-koden kommer av at den kan leses raskt uavhengig av rotasjon og har kapasitet på over 7000 tegn. QR-koden ble lansert uten beskyttelse og er tilgjengelig for alle å bruke [4].



Figur 2. QR-kode som inneholder teksten «http://www.QRcode.com/». De fire hjørnene blir funnet av ZBar i rekkefølgen 0 til 3, uavhengig av hvilken orientering den har i bildet. Orienteringen og posisjonen blir bestemt av de karakteristiske rektanglene i tre av hjørnene. Resten av QR-koden inneholder den lesbare datastrengen.

Som vist i Figur 2, består QR-koden av tre punkter for posisjon og orienteringsbestemmelse. De resterende delen kalles moduler. QR-koden har størrelse avhengig av mengden informasjon som er lagret i koden [4]. Figur 2 er en 25x25 modul QR-kode, som betyr at den har 25 svarte eller hvite firkanter horisontalt og vertikalt. Her kan det kodes inn 47 alfanumeriske tegn i datastrengen.

QR-koden har en funksjon for å korrigere feil ved for eksempel skade på QR-koden, eller at noen av modulene er dekket til. Når QR-koder genereres, bestemmes feilkorrigeringen ved fire nivåer hvor

det blir godtatt at henholdsvis 7, 15, 25 eller 30% av dataene mangler ved lesing. Ved å øke nivået for feilhåndtering, går forholdet mellom lagringskapasitet og moduler ned [4].

QR-matrise

QR-matrise i dette prosjektet er en samling QR-koder som er plassert i et rutenett slik som Figur 1 viser, en hel QR-matrise ligger som vedlegg i kap. 7. Hver av QR-kodenes datastreng inneholder plasseringen til det nederste venstre hjørnet av QR-koden i romkoordinater. Den inneholder også den fysiske størrelsen til QR-koden. QR-matrisen er produsert med et pythonprogram som tidligere er laget ved Universitet i Stavanger [5].

QR-kodeleser

ZBar Bar Code Reader Library er et bibliotek som ligger åpent og tilgjengelig på internett [6]. Biblioteket er skrevet i C++ og har tre nivåer av grensesnitt for å lese strekkoder.

Høgnivå:
Dette nivået inneholder de mest nødvendige metodene for å lese strekkoder fra en videostrøm og er satt opp for å være lett i bruk. Det er også en Python-wrapper til dette nivået [6].

Mellomnivå:
Dette nivået gir tilgang til det som kalles bildeobjekt, eller «Image». Dette objektet inneholder strekkoder som er funnet ved scanning. Strekkodene er organisert i en iterator, slik at det er enkelt å iterere seg gjennom resultatene. Fra hvert resultat kan det hentes ut x- og y-koordinater for hvert av hjørnene i tillegg til datastrengen som er lagret i strekkoden. Nivået inneholder også metoder å for laste inn bilder fra en videostrøm, og for å vise resultater på skjermen [6].

Lavnivå:
Dette nivået gir tilgang til rådataene fra et bilde. Strekkodene blir detektert basert på kanter, retninger og avstander. Dette er det mulig å påvirke i lavnivået [6].

QR-koder er en av flere typer strekkoder som kan leses med ZBar. Videre i dette prosjektet blir mellomnivået benyttet for å lese x- og y-koordinatene til hvert av hjørnene i QR-koden, samt å lese datastrengen som er lagret i QR-koden. Hjørnekoordinatene blir lest i rekkefølgen 0 til 3, som Figur 2 viser, uavhengig av hvilken orientering den har i bildet. Dette er nyttig blant annet for å se om bildet er tatt opp ned eller riktig vei.

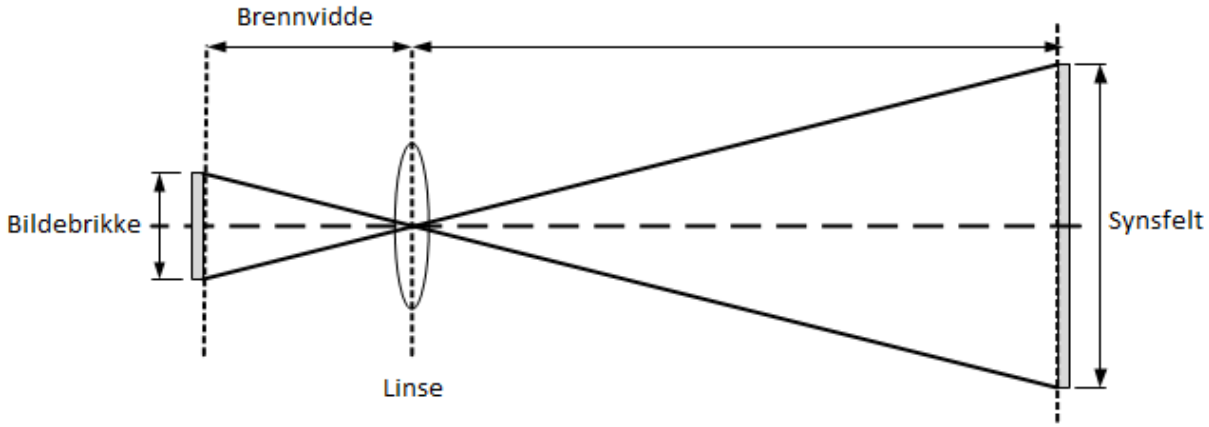
2.2 Kamerateori

2.2.1 Kamera

I dette prosjektet brukes digitale industrikameraer med utskiftbare objektiver. Kameraet består av et kamerahus med tilhørende elektronikk og en bildebrikke. Bildebrikken er en rektangulær matrise av lyssensorer som sitter inne i kamerahuset. Hver lyssensor måler et nivå av lysstyrke og definerer verdien til en piksel. Oppløsningen til bildet fra et kamera er identisk med oppløsningen av sensorer på bildebrikken [7].

Objektivet sin oppgave er å samle lys fra det som er ønsket synsfelt og sende det inn på bildebrikken. Et objektiv er som regel bygd opp av flere linser som står etter hverandre. Brennvidden definerer synsfeltet til kameraet, se Figur 3. En korter brennvidde vil gi et større synsfelt og en lengre vil gi et mindre synsfelt. Et objektiv med stort synsfelt kalles vidvinkelobjektiv. Et objektiv med smalt synsfelt kalles teleobjektiv [8].

Det som kalles et normalobjektiv er et objektiv som har omtrent likt synsfelt som menneskets øye. Normalobjektivet har brennvidde ved det punktet som skiller vidvinkel og teleobjektiv. Det er vanskelig å lage et objektiv som tilsvarer det teoretisk perfekte normalobjektivet. Objektiv med brennvidden lik diagonalen av bildebrikken er et godt utgangspunkt for å finne normalobjektivet til kameraet [9].



Figur 3. Todimensjonal prinsippskisse for kamera. Her er objektivet illustrert med en enkel linse. Ved å øke brennvidden, vil synsfeltet bli mindre og motsatt.

Det er mulig å regne ut størrelsen på synsfeltet til kameraet ved å ta utgangspunkt i noen parametere. H_b og W_b er høyde og bredde i bildet oppgitt i piksler. I tillegg til høyden og bredden, trengs p, som er pikselstørrelsen på bildebrikken og f, som er brennvidden til objektivet. Alt dette kan hentes fra datablad til kamera og objektiv. Det er ønskelig finne H_r og W_r som er henholdsvis høyde og bredde til synsfeltet oppgitt i millimeter.

For å finne høyden og bredden brukes det faktumet at forholdet mellom brennvidden og størrelsen på bildebrikken er det samme som forholdet mellom avstanden til motivet (d), og størrelsen på synsfeltet. Figur 3 kan være til hjelp for å forstå prinsippet. Ligning (1) og (2) gir da høyden og bredden på synsfeltet.

$$\frac{H_r}{d} = \frac{p \cdot H_b}{f} \quad \rightarrow \quad H_r = \frac{p \cdot H_b \cdot d}{f} \tag{1}$$

$$\frac{w_r}{d} = \frac{p \cdot w_b}{f} \quad \rightarrow \quad w_r = \frac{p \cdot w_b \cdot d}{f} \tag{2}$$

2.2.2 Linseforvrengning

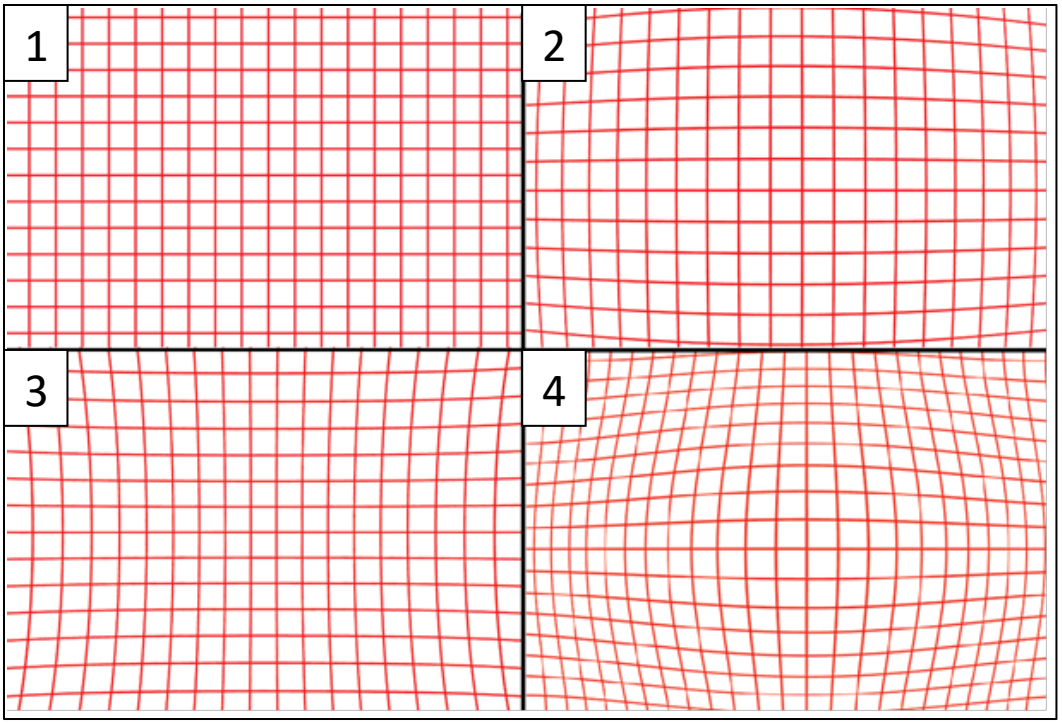
Radial linseforvrengning av bildet oppstår når et bilde blir tatt gjennom et objektiv som gir et synsfelt som ikke har nøyaktig den rektangulære formen som bildebrikken har [10].

Det blir som regel skilt mellom to hovedkategorier av radial forvrengning, negativ og positiv. Her blir de betegnet som tønneforvrengning og puteforvrengning. Det er også en tredje type som er en kombinasjon av disse to. Graden av den ene eller andre typen forvrengning avhenger av hvilken type objektiv kameraet har. En grei måte å se nivået av forvrengningen på, er å se på et bilde av det som i virkeligheten skal være rette linjer. Desto mer forvrengning det er, desto mer vil slike linjer bli avbøyd på bildet [10].

Tønneforvrengning
Denne typen forvrengning oppstår når det blir tatt bilde med et vidvinkelobjektiv [10]. Figur 4 viser et rutenett der nr. 1 er uten forvrengning. Alle linjene er rette. Nr. 2 er et eksempel på tønneforvrengning. Her er linjene bøyd innover i kantene, og motivet får en tønneform. Bildet blir presset sammen i kantene.

Dette fenomenet oppstår fordi en økning av synsvinkelen tilsvarer en større økning av avstand i ytterkantene av synsfeltet enn hva den gjør rundt senter av synsfeltet. Siden lyset blir sendt inn på en bildebrikke som har like store piksler over hele bildekanten, vil en piksel representere en større del av motivet i hjørnene enn hva den vil i de sentrale delene av bildet [10].

Puteforvrengning
Dette oppstår på motsatte premisser av tønneforvrengning. Når objektivet har lite synsfelt, vil en piksel i hjørnet representere en mindre avstand enn en piksel midt på bildekanten. Motivet blir da sterkt ut mot hjørnene [10]. Nr. 3 i Figur 4 vises puteforvrengning. Legg merke til at selv om det er forvrengning i bildene, så er linjene rette i områdene i senter av bildet. Nederst til høgre i Figur 4 viser det som kalles mustasjeforvrengning som er en kombinasjon av pute- og tønneforvrengning [10].



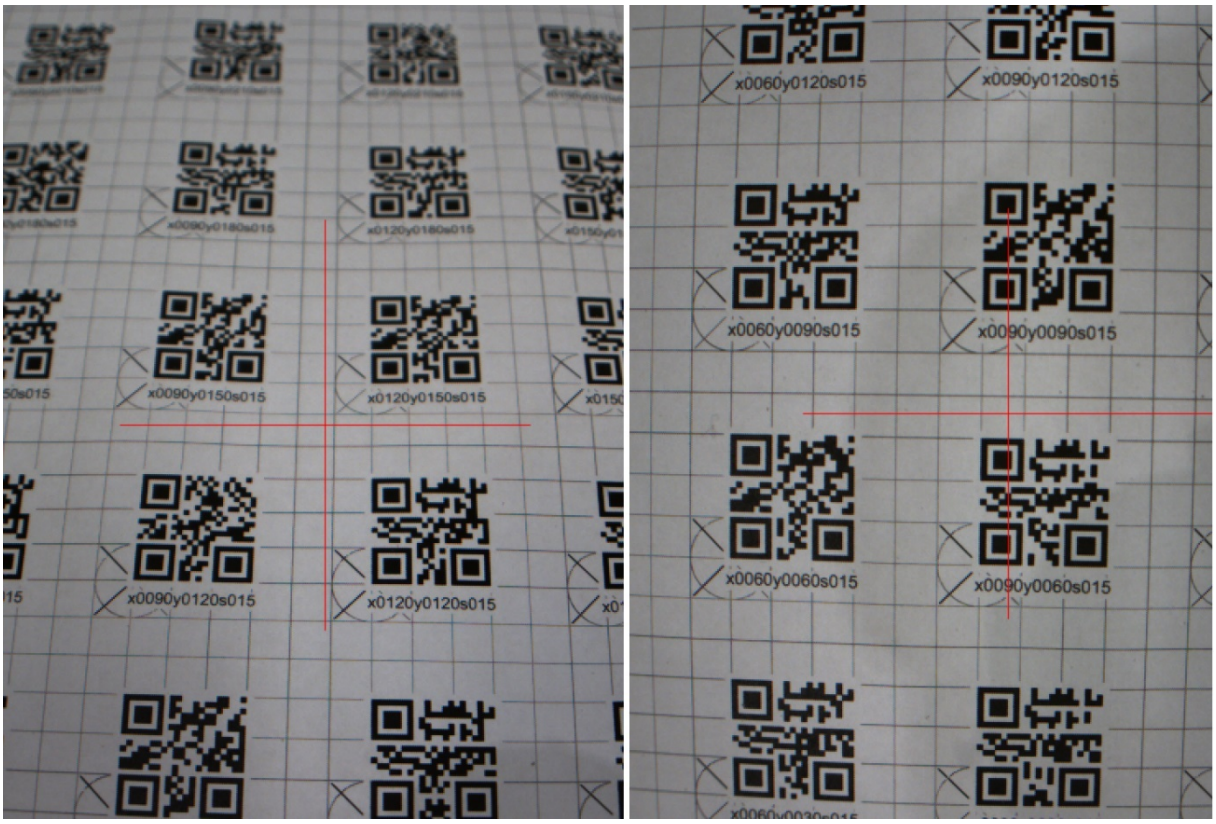
Figur 4. Rutenettet illustrerer hva som skjer med rette linjer når de blir påvirket av radial linseforvrengning. Legg merke til at linjene er rette i senter av bildene. Rute 1: uten forvrengning, rute 2: tønneforvrengning, rute 3: puteforvrengning, rute 4: Kombinasjon av pute- og tønneforvrengning. De fire bildene er hentet fra [10].

2.2.3 Perspektivforvrengning

Perspektivforvrengning handler om å plassere den tredimensjonale verden på et todimensjonalt bilde. Avstanden fra kameraet til detaljer i motivet avgjør hvor store de blir i bildet. Om et objekt i motivet har stor avstand til kameraet, vil de se mindre ut enn om avstanden er mindre [10]. Dette er akkurat det samme som skjer med det menneskelige synet. Om personen som ser står helt inntil et hus, kan det se stort ut i forhold til et fjell i bakgrunnen. Dette er et naturlig fenomen, men om det ikke blir tatt med i betraktningen kan det forårsake feil ved bruk av maskinsyn til målinger.

En måte å løse problemet på er å bruke et teleobjektiv og ta bildet fra en større avstand. Avstanden mellom objektene er fortsatt like stor, men avstandene fra kameraet til hvert av objektene er mye likere hverandre [10]. Dette kan sammenlignes med å gå langt fra huset og se på det med kikkert. Huset vil da se mye mindre ut enn fjellet i bakgrunnen, slik som det i virkeligheten er.

I dette prosjektet blir det tatt bilder av en QR-matrise som er plan, og kameraet står normalt på planet. Det vil da bli minimalt eller ingen perspektivforvrengning i bildet. Ved å endre vinkelen til kameraet, vil perspektivforvrengning påvirke resultatet. Denne effekten viser Figur 5.



Figur 5. Til venstre er kameraet tiltet 25°, og dette viser effekten av perspektivforvrengning. QR-kodene nederst i bildet er nærmere kameraet og ser større ut enn de øverst i bildet. Det høyre bildet er tatt med 0° tilt, og her har alle QR-kodene den samme dybden og de ser like store ut. Figuren viser også at det blir et ulikt fokusplan i bildet når kameraet er tiltet.

2.2.4 Kamerakalibrering

Kamerakalibrering i denne sammenhengen består av å finne parametere for kamera for å kunne kompensere for radial linseforvrengning, som er forklart i kap.2.2.2. Det må også kompenseres for tangential forvrengning som oppstår når bildebrikken ikke står parallelt bak objektivet. Dette vil resultere i at motivet vil bli større i det områdene hvor bildebrikken er nærmere objektivet og motsatt [11]. Her snakkes det om kamera som en kombinasjon av bildebrikke og ikke justerbart objektiv da et zoomobjektiv vil ha forskjellige parametere avhengig av zoomnivå.

Ligningene (3) og (4) beskriver radial linseforvrengningen. La x_d og y_d være de forvrengte koordinatene. x og y er de ikke-vrengte koordinatene og $r^2 = x^2 + y^2$. Koordinatene er normalisert ved at de er sentrerte og dividert på fokallengden [11, 12].

$$x_d = x(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \quad (3)$$

$$y_d = y(1 + k_1 \cdot r^2 + k_2 \cdot r^4 + k_3 \cdot r^6) \quad (4)$$

Ligningene (5) og (6) beskriver translasjonen i tangential forvrengning [11] [13].

$$x_d = x + (2 \cdot p_1 \cdot x \cdot y + p_2 \cdot (r^2 + 2 \cdot x^2)) \quad (5)$$

$$y_d = y + (2 \cdot p_2 \cdot x \cdot y + p_1 \cdot (r^2 + 2 \cdot y^2)) \quad (6)$$

Det er fem parametere eller forvrengningskoeffisienter som må finnes *(k_1, k_2, p_1, p_2 og k_3)*. I de fleste tilfeller blir det sett bort fra k_3, da den først har innvirkning ved stor forvrengning [11]. Ved interesse, kan det leses mer om bakgrunnen for likningene (3-6) i blant annet *A Flexible New Technique for Camera Calibration* [14].

Kameramatrisen er en 3x3 matrise som inneholder noen parametere for kameraet. Den er satt opp som i (7). c_x og c_y er fokalpunktet i bildet. f_x og f_y er fokallengden, eller brennvidden til kameraet. Parametere α utrykker skjevheten mellom x og y -akse [14].

$$\begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (7)$$

Det finnes flere måter å komme frem til forvrengningskoeffisientene og kameramatrisen på. I denne rapporten blir en metode som baserer seg på å ta flere bilder av et sjakkbrett, med kjent størrelse, fra forskjellige vinkler fulgt. Dette er en kjent metode som er implementert i både Matlab [15] og OpenCV [12]. Implementasjonene er basert på *A Flexible New Technique for Camera Calibration* [14].

For å kalibrere kamera med denne algoritmen i OpenCV, trengs det bilder av et sjakkbrett fra mange ulike vinkler. Ved å lokalisere hjørner mellom hvite og svarte rektangler i bildene setter algoritmen opp et ligningssystem for å bestemme forvrengningskoeffisientene og kameramatrisen. Disse parameterne blir normalt lagret og brukes til å rette forvrengte bilder tatt med det aktuelle kameraet. Det blir da ikke nødvendig å kalibrere kameraet på nytt såfremt optikken ikke blir endret. Om oppløsningen på kameraet endres, må kameramatrisen skaleres deretter [12].

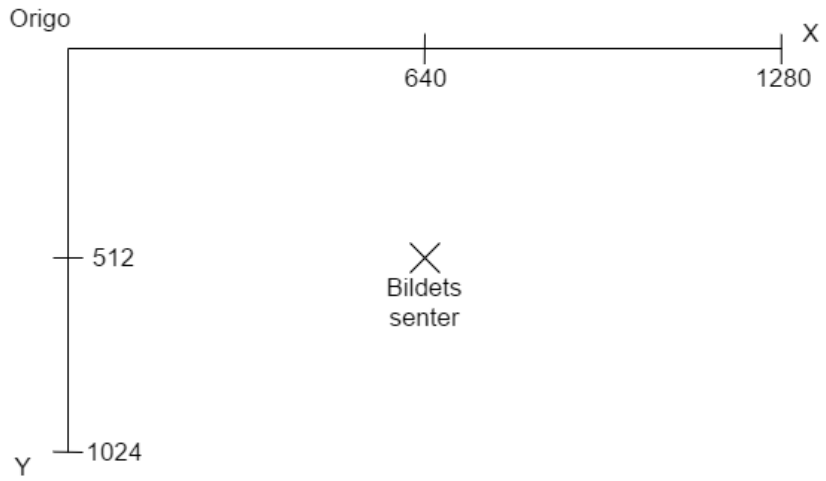
For å rette forvrengte bilder i OpenCV, bruks en av to metoder, «undisorte», eller «undisortPoint». Den første trenger kameramatrisen og forvrengningskoeffisientene som input i tillegg til bildet. Metoden bruker ligningene (1-4) for å organisere alle pikslene i bildet slik at forvrengningen fjernes [16]. Den andre metoden tar ikke hele bildet som inngang, men en vektor av de punktene som er ønskelig å korrigere. Når metodene kjøres må det også gjøres et valg for å få plass til de nye pikslene. Bildets oppløsning må økes, bildets kanter må kuttes, eller bildet må komprimeres.

2.3 Geometri i bildet

Dette kapittelet gir en lett introduksjon eller repetisjon til noen geometriske forhold som blir brukt videre i rapporten.

Koordinatsystemet

Koordinatsystemet i et bilde blir i denne rapporten kalt for bildekoordinatsystem, og det har origo i øverst venstre hjørne. Et eksempel er vist i Figur 6. Oppløsningen i koordinatsystemet er pikslene i bildet. Vanligvis blir oppløsningen til et bilde oppgitt som for eksempel 1280x1024, i dette tilfelle har bildet 1280 piksler horisontalt og da går X-aksen fra 0 til 1280. Det er 1024 piksler vertikalt, og Y-aksen går fra 0 til 1024. Senter av bildet finnes som oppløsningen horisontalt og vertikalt delt på to.



Figur 6. Koordinatsystemet til et bide med oppløsning 1280x1020.

Rotasjon av et bilde

I mange tilfeller kan det være nødvendig å rotere et bilde. Dette er trivielt å gjøre om vinkelen bildet skal roteres med er kjent. Her blir metoden for å rotere en piksel forklart. Skal hele bildet roteres, så må denne operasjonen gjøres for hver piksel.

For å rotere et punkt i bildet, flyttes de aktuelle punktet ved hjelp noen ligninger. Ligningene kan skrives på matriseform med rotasjonsmatrisen som i (8). De kan også skrives oppdelt som i (9) og (10) for lettere implementering i c++ eller andre programmeringsspråk [17, 18]. I innledningen til kap. 3 er det gjort ett talleksempel av en rotasjon med disse ligningene.

Det er ønskelig å rotere punktet P rundt senter av bildet. Det er da nødvendig å referere punktet til senter istedenfor origo. La derfor R være P - C. der C er senterpunktet til bildet. La R’ være de roterte koordinatene med referanse til senter.

$$\begin{bmatrix} R_x' \\ R_y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} R_x \\ R_y \end{bmatrix} \quad (8)$$

$$R_x' = R_x \cos\alpha + R_y \sin\alpha \quad (9)$$

$$R_y' = R_y \cos\alpha - R_x \sin\alpha \quad (10)$$

Punktet R' er referert til senter av bildet, P' er det roterte punktet referert til origo av bildet og finnes ved å addere senterpunktet.

P'= R'+C (11)

Vektorregning

I dette prosjektet regnes det mye med piksler i et todimensjonalt bilde. Det kan være greit å se på posisjonene i bildet, forhold og linjer mellom dem som vektorer. Her kommer noen ligninger som er greit å ha med videre:

Vektoren V fra et punkt a til et annet punkt b:

$$V = b - a \quad (12)$$

Det kan deles videre i:

$$V_x = b_x - a_x \quad (13)$$

$$V_y = b_y - a_y \quad (14)$$

Lengden l av en vektor V:

$$l = \sqrt{V_x^2 + V_y^2} \quad (15)$$

Vinkelen α til en vektor V:

$$\alpha = arctan(V_y/V_x) \quad (16)$$

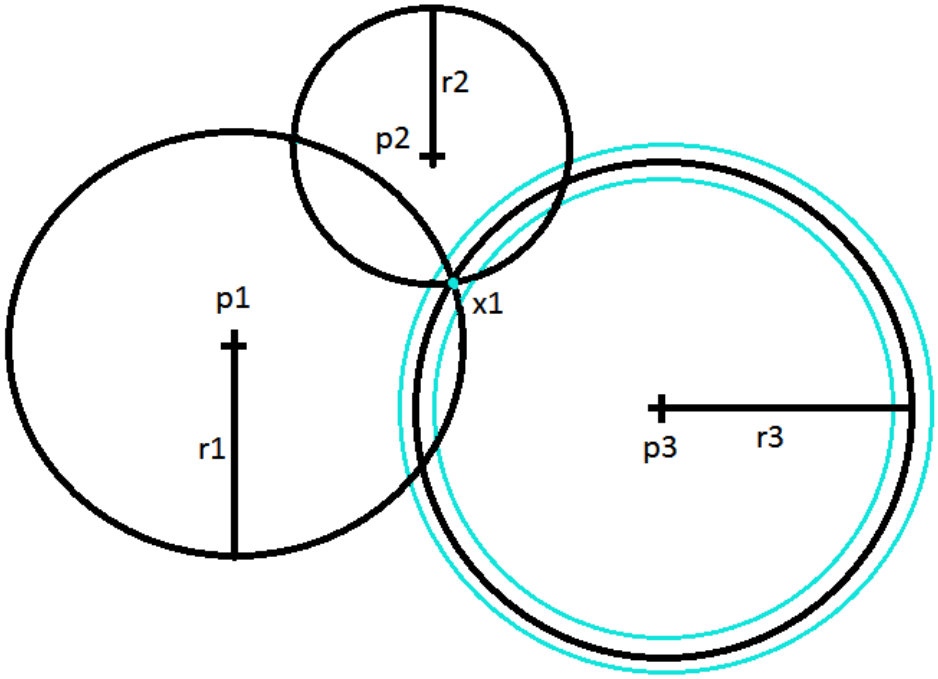
2.4 Trilaterasjon

Trilaterasjon handler om å bestemme posisjonen til et ukjent punkt ved å bruke punktets avstand til flere andre kjente punkter. På et todimensjonalt plan trengs det minimum tre kjente punkter og avstanden deres til det ukjente punktet. Det ukjente punktet eksisterer da i den posisjonen der sirkler med radius lik avstandene fra de kjente punktene krysser hverandre [19].

La p1, p2 og p3 være kjente punkter, og r1, r2 og r3 være deres avstander til det ukjente punktet x1, illustrert med de sorte linjene i Figur 7. Om punkt p3, eller et av de andre punktene, ikke hadde vert tilgjengelig, og uten andre antagelser, hadde det vert to like sannsynlige krysningspunkter. Det tredje punktet gjør at det blir bare en løsning.

Trilaterasjon er grunnprinsippet i mange navigasjonssystemer. Avstandene kan peiles fra satellitter (GPS) eller basestasjoner som for eksempel mobilmaster. Da brukes sendetiden og kjent fart på signalet til å estimere avstanden eller radiusen. I det tredimensjonale rommet brukes det samme prinsippet som i 2D, men her brukes det kuler som skjærer hverandre i rommet [19].

Det kan lett tenkes at det ved slike systemer i praksis er noe målefeil i radiusen. Da vil slike kuler eller sirkler ikke nødvendigvis ha et perfekt skjæringspunkt. La r3 i Figur 7 være litt større eller litt mindre, slik som de blå sirklene indikerer. Skjæringspunktet x1 eksisterer ikke lenger, og det finnes ikke et felles skjæringspunkt for alle de tre sirklene.



Figur 7. 2D-prinsipp for trilaterasjon. Det ukjente punktet x1 bestemmes av krysningspunktet til sirkler med radius lik avstanden mellom kjent punkt, p og ukjent punkt, x. De blå sirklene viser hva som skjer om radiusen ikke er perfekt målt. Det finnes ikke lenger et krysningspunkt for alle tre sirklene.

For å ta hensyn til målefeil i radiene, må et krysningspunkt tilpasses. En måte å gjøre dette på er minste kvadraters metode, eller Least Square. Denne metoden handler om å finne det punktet som har den minste kvadrerte avstanden til hver av sirklene.

Ligningen for sirkel i kan skrives som (17) der px og py er henholdsvis x- og y-koordinaten til senterpunktet. X og y er et vilkårlig punkt på sirkelen, r er radiusen [20, 21].

$$(\mathrm{x} - \mathrm{px}_i)^2 + (\mathrm{y} - \mathrm{py}_i)^2 = \mathrm{r}_i^2 \tag{17}$$

Med *k* antall sirkler er det *k* antall ligninger i systemet.

Dette er ulineære ligninger, og de kan derav ikke løses direkte. En mulig fremgangsmåte er å lineærisere disse ligningene. Ved å bruke den første ligningen som et lineæriseringsverktøy, er resultatet ligning (18), etter at p_1 er lagt til og trukke fra i alle ledd [20, 21].

$$(\mathrm{x} + \mathrm{px}_1 - \mathrm{px}_1 - \mathrm{px}_{i+1})^2 + (\mathrm{y} + \mathrm{py}_1 - \mathrm{py}_1 - \mathrm{py}_{i+1})^2 = \mathrm{r}_{i+1}^2 \tag{18}$$

r_1 er avstanden mellom det ukjente punktet og kjent punkt fra den første ligningen. r_{i+1} er avstanden mellom det ukjente punktet og den aktuelle ligningens punkt. r_1-r_{i+1} er avstanden, mellom punkt 1 og den aktuelle ligningens punkt. Dette kan videre skrives som:

$$(\mathrm{x} - \mathrm{px}_1)(\mathrm{px}_{i+1} - \mathrm{px}_1) + (\mathrm{y} - \mathrm{py}_1)(\mathrm{py}_{i+1} - \mathrm{py}_1) = \frac{1}{2}(\mathrm{r}_1^2 - \mathrm{r}_{i+1}^2 + (r_1 - r_{i+1})^2) \tag{19}$$

Det er nå n-1 lineære ligninger slik som ligning (19) med 2 ukjente, x og y. Dette kan skrives på matriseform:

$$\mathrm{Aq} = b \tag{20}$$

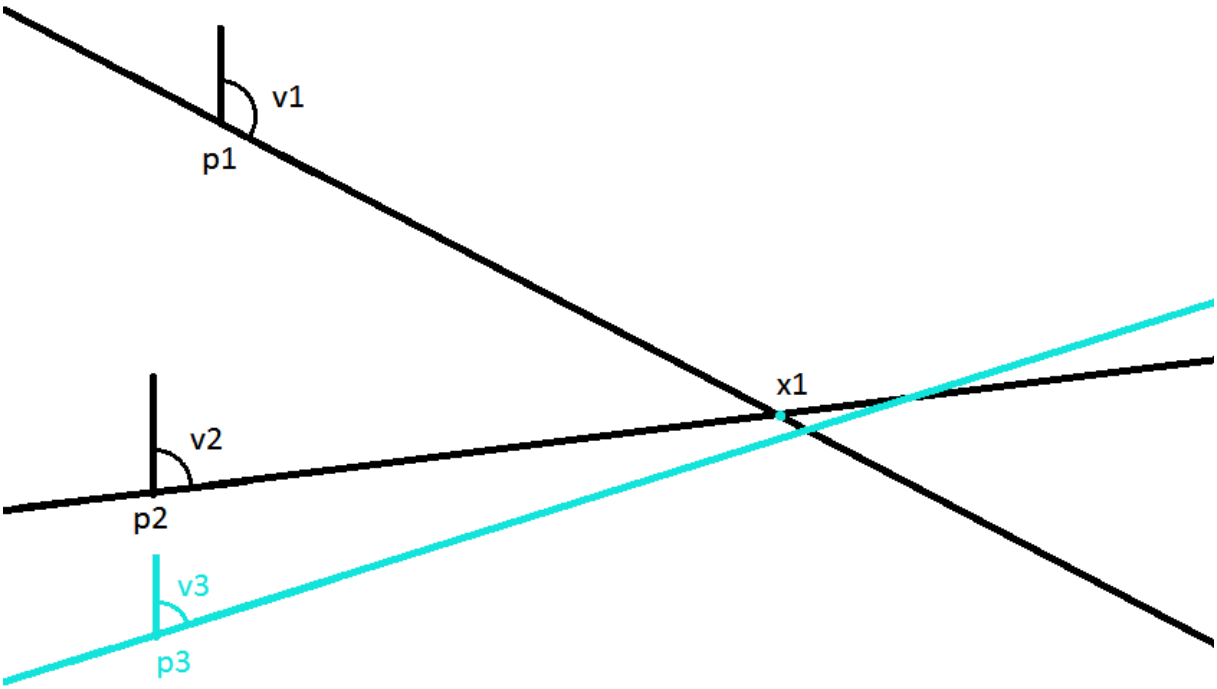
$$\mathrm{A} = \begin{bmatrix} (\mathrm{px}_2 - \mathrm{px}_1) & (\mathrm{py}_2 - \mathrm{py}_1) \\ (\mathrm{px}_3 - \mathrm{px}_1) & (\mathrm{py}_3 - \mathrm{py}_1) \\ \vdots & \vdots \\ (\mathrm{px}_k - \mathrm{px}_1) & (\mathrm{py}_k - \mathrm{py}_1) \end{bmatrix}, \quad \mathrm{q} = \begin{bmatrix} x - x_1 \\ y - y_1 \end{bmatrix}, \quad b = \begin{bmatrix} \frac{1}{2}(\mathrm{r}_1^2 - \mathrm{r}_2^2 + (r_1 - r_2)^2) \\ \frac{1}{2}(\mathrm{r}_1^2 - \mathrm{r}_3^2 + (r_1 - r_3)^2) \\ \vdots \\ \frac{1}{2}(\mathrm{r}_1^2 - \mathrm{r}_k^2 + (r_1 - r_k)^2) \end{bmatrix}$$

Ved å bruke Singular Value Decomposition(SVD), kan den inverterte matrisen A^{-1} finnes, og løsningen q finnes som (21). Å løse dette ligningsettet med SVD, eller andre dekomposisjonsmetoder, kan gjøres enkelt med metoden Solve fra OpenCV [22], eller andre verktøy.

$$\mathrm{q} = A^{-1}b \tag{21}$$

2.5 Triangulering

Triangulering handler om å se på vinkler og trekanter i geometrien for å lokalisere punkter. I Figur 8, la p1 og p2 være kjente punkter på 2D planet. V1 og v2 er deres respektive vinkler mellom den vertikale aksen og linjen til det ukjente punktet x1. Ved å kjenne punktene og deres vinkler, eller retningsvektorer, kan det lages linjer fra pluss uendelig til minus uendelig. Det ukjente punktet x1 finnes der disse linjene krysser hverandre.



Figur 8. To eller flere punkter er kjent. Deres vinkel til det ukjente punket x1 er også kjent. Krysspunket til disse linjene er den ukjente x1. Den blå linjen indikerer hva som skjer om det er målefeil i systemer med mer enn to punkter. Det finnes ingen felles krysningspunkt.

Om det er tre eller flere linjer som bestemmer x1, oppstår det samme problemet som for trilaterasjon i kap. 2.4. Målefeil i måleinstrumentet som finner vinkelen, eller feil i p-koordinatene, gjør at det ikke blir noen felles krysningspunkt for alle linjene. Den blå linjen i Figur 8 indikerer en målt vinkel med litt feil.

På samme måte som i kap. 2.4 brukes minste kvadraters metode for å finne x1 som har den minste kvadratiske avstand til linjene. Ved å bytte ut vinkelen med en normalisert retningsvektor n, med lengde 1, er ligningen for linjen [23]:

$$x = p + t\,n \tag{22}$$

der:

$$\mathrm{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \quad n = \begin{bmatrix} n_x \\ n_y \end{bmatrix}, \quad -\infty < t < \infty$$

Den vinkelrette linjen fra et punkt til linjen kan videre beskrives som (23) der I er identitestmatrisen [23].

$$l = (\mathrm{p} - \mathrm{x})^T (\mathrm{I} - \mathrm{nn}^T)\ (\mathrm{p} - \mathrm{x}) \tag{23}$$

Ligning (23) kan deriveres med hensyn på x og settes lik 0:

$$-2(\mathrm{I} - \mathrm{nn}^T)\ (\mathrm{p} - \mathrm{x}) = 0 \tag{24}$$

Disse ligningene er lineære og kan settes opp i et system for å løses. Med k antall linjer i bildet blir det k antall ligninger. Ved å organisere ligningene, blir ligningssystemet:

$$\mathrm{Rp} = \mathrm{q} \tag{25}$$

$$\mathrm{R} = \begin{bmatrix} (\mathrm{I} - \mathrm{n}_1\mathrm{n}_1{}^T) \\ (\mathrm{I} - \mathrm{n}_2\mathrm{n}_2{}^T) \\ \vdots \\ (\mathrm{I} - \mathrm{n}_k\mathrm{n}_k{}^T) \end{bmatrix}, \ \mathrm{x} = \begin{bmatrix} x_x \\ x_y \end{bmatrix}, \ q = \begin{bmatrix} (\mathrm{I} - \mathrm{n}_1\mathrm{n}_1{}^T) \cdot \mathrm{p}_1 \\ (\mathrm{I} - \mathrm{n}_2\mathrm{n}_2{}^T) \cdot \mathrm{p}_2 \\ \vdots \\ (\mathrm{I} - \mathrm{n}_k\mathrm{n}_k{}^T) \cdot \mathrm{p}_k \end{bmatrix},$$

Dette ligningsettet løses med SVD på lik måte som i kap. 2.4

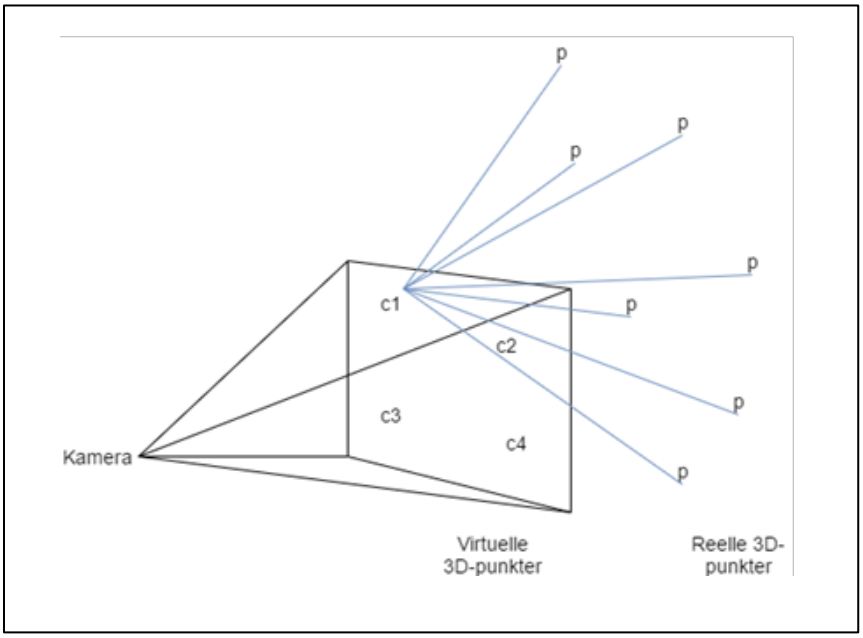
2.6 OpenCV

OpenCV er et bibliotek med over 2500 algoritmer og metoder for bildehandling og maskinsyn. Det er utviklet under BSD-lisens som betyr at det er gratis å bruke for alle. Biblioteket har grensesnitt for C++, C, Python, Java og Matlab. Det er i stor grad utviklet og optimalisert for effektiv implementasjon og real time applikasjoner [24].

2.7 EPnP

Målet med Perspective-n Point Problemet (PnP), som ble introdusert i kap. 1.3, er å estimere et kamera sin posisjon i rommet basert på punkter som er kjent både i bildets koordinatsystem og i rommets tredimensjonale koordinatsystem. I 2008 ble det utviklet en ikke-iterativ løsning av problemet som har gitt både effektive og presise resultater, *EPnP: An Accurate O(n) Solution to the PnP Problem* [25].

Algoritmen er basert på fire virtuelle referansepunkter som er plassert på optimale steder i rommet. For hvert av disse punktene definertes en vektet sum av alle de tredimensjonale referansepunktene som er tilgjengelige. Figur 9 kan være til nytte for å forstå prinsippet. På denne måten gjør EPnP nytte av presisjonen til mange punkter i rommet, men regnekompleksiteten videre er redusert til å estimere 4 punkter i rommet [25].



Figur 9. Prinsippskisse for EPnP. Punktene p er de reelle 3D-punktene som er kjent i rommet. De fire virtuelle punktene c1-c4 ligger på forhåndsbestemte og optimale plasseringer i rommet. En vektet sum av alle p-punktene for hver c blir brukt videre i beregningene. De svarte strekene i figuren indikerer synsfeltet til kameraet frem til c-punktene. De blå strekene indikerer vektene til p-punktene for punk c1.

Det kan vises at de vektede summene, punktene i bildet og den interne kalibreringsmatrisen til kameraet kan settets opp og forenkles til en 12x12 matrise. Denne matrisens nullrom gir løsningen av kameraets posisjon i romkoordinater [25].

Videre i dette prosjektet brukes EPnP som en algoritme for å finne senterpunktet i bildet. I de tilfellene kameraet står normalt mot punktenes koordinatsystem vil kameraet sine x- og y-koordinater stemme med bildets senterpunkt. EPnP er en algoritme som er implementert i OpenCV og brukes direkte derfra [26].

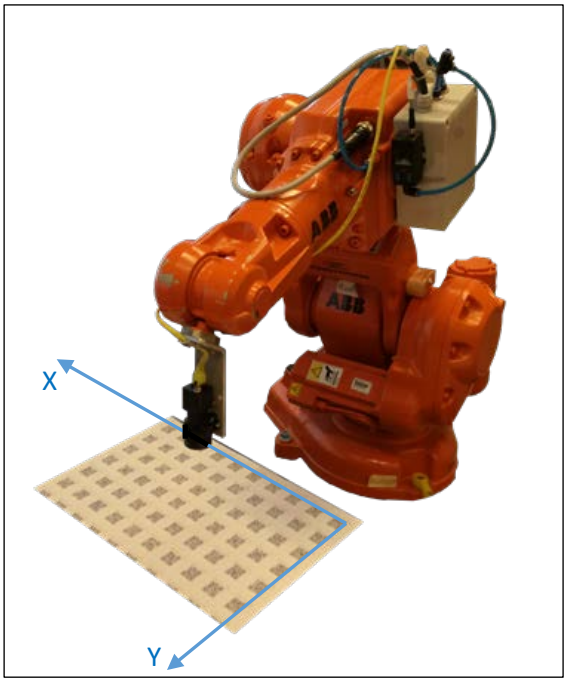
3. Algoritmer som er utviklet i dette prosjektet

Formålet med algoritmene som er utviklet i dette prosjektet er å lokalisere senter av et bilde i et eksternt koordinatsystem. Det som menes med senter av bildet er den pikselen som ligger midt på x-aksen og midt på y-aksen til bildet. Dette punktet er markert med et rødt kryss i Figur 10

Med et eksternt koordinatsystem menes et koordinatsystem som står fast i rommet. I Figur 11 er dette koordinatsystemet tegnet inn, og videre blir det referert til dette koordinatsystemet som romkoordinatsystem og romkoordinater. Figur 11 viser også en robot som holder et kamera. Figur 10 viser synsfeltet til dette kameraet, og her er bildets koordinatsystem tegnet inn.

Problemet som må løses er å finne ut hvor senter av bildet, rødt kryss i Figur 10, er oppgitt i romkoordinatsystemet. Nøkkelen til å løse dette er en matrise av QR-koder som ligger i, eller delvis i synsfeltet til kameraet. QR-matrisen vises både i Figur 11 og Figur 10. For enkelhets skyld er QR-matrisen plassert i origo av romkoordinatsystemet, men dette er ikke en betingelse.

Datastrengen som kan leses fra hver av QR-kodene inneholder x- og y-koordinaten til det nedre venstre hjørnet av QR-koden, oppgitt i romkoordinater. Strengen inneholder også lengden av sidene til QR-koden. Med forbehold om at QR-matrisen er plassert med kjent vinkel til romkoordinatsystemet, kan alle fire hjørnenes posisjon enkelt regnes ut. I dette prosjektet er QR-matrisen plassert med positiv x- og y-retning fra origo, og med 0° vinkel til romkoordinatsystemet slik som Figur 11 viser.



Figur 11. Eksternt koordinatsystem, eller romkoordinatsystem. Dette koordinatsystemet står fast i rommet uavhengig av hvor kameraets synsfelt er. Her er en QR-matrise plassert i origo av koordinatsystemet.

Figur 10. Bildets koordinatsystem. Dette er det bildet kameraet ser, koordinatsystemet er pikselverdiene i x- og y-retning, og det røde krysset indikerer senterpunktet til bildet. Kameraet ser en QR-matrise, det er QR-kodene som er interessante for systemet, rutenettet i bakgrunnen er bare for visuell inspeksjon.

Rutenettet i bakgrunnen, bokstaver og tall som er skrevet i QR-matrisen er bare for visuell inspeksjon. Det kan være til hjelp for å forstå prinsippet, men algoritmene som er utviklet videre i prosjektet gjør ikke nytte av annet enn QR-kodene. Ved å bruke synsfeltet i Figur 10 som eksempel, har den QR-koden som ligger til venstre for senter, datastrengen: «x0050y0100s025». Av dette kan det trekkes ut at QR-kodens hjørne nederst til venstre er 50 mm i x-retning og 100mm i y-retning fra origo av romkoordinatsystemet, det skrives lettere: (50,100). Størrelsen på QR-koden er 25x25 mm, og QR-matrisen har som nevnt 0° vinkel i forhold til romkoordinatsystemet. Det kan da regnes frem til de andre tre hjørnenes romkoordinater. For eksempel blir romkoordinaten til hjørnet øverst til høyre (75,125).

Det to parameterne, pikselstørrelse og rotasjon rundt z-aksen må finnes fra bildet. Disse to parameterne blir brukt videre til å regne seg frem til bildesenteret sine koordinater i romkoordinatsystemet. For å måle avstander, regnes pikselstørrelsen ut. Det som menes med pikselstørrelse i dette prosjektet, er hvor mange millimeter i romkoordinater en piksel av bildet representerer. Om for eksempel bredden av en QR-kode i Figur 10 måles opp til å være 250 piksler i bildet, og fra datastrengen kan det leses at den er 25 mm bred, så er pikselstørrelsen 0,1 mm. Rotasjonen rundt z-aksen til kameraet, eller med andre ord vinkelen mellom koordinatsystemene, må også finnes. I Figur 10 ser denne vinkelen ut til å være ca. 10°. Både pikselstørrelsen og rotasjonen kan finnes på ulike måter, og dette er forklart nærmere i kap. 3.1 og 3.2.

Når et bilde skannes med ZBar, leses datastrengen til hver QR-kode i bildet, og i tillegg lagres alle hjørnene til hver QR-kode i bildekoordinater. Bildekoordinatene er plasseringen i bildet i henhold til koordinatsystemet som er tegnet inn i Figur 10. Systemet kjenner da bildekoordinatene til QR-kodens hjørner og til bildets senter som alltid er fast i bildekoordinatsystemet.

Det er ulike måter å bruke disse parameterne på for å finne bildets senterpunkt i romkoordinater. De algoritmene som er utviklet i dette prosjektet er detaljert forklart i kap. 3.3. Den kanskje mest opplagte måten å gjøre dette på er den algoritmen som videre blir referert til som Enkelmetode. Hovedprinsippet blir forklart her ved et talleksempel i Eksempel 1:

Eksempel 1. Hovedprinsippet for å finne senterposisjon av bildet med Enkelmetode

- Funnet pikselstørrelse fra bildet: 0,1 mm
- Funnet rotasjon om z-aksen: -10°
- Ett funnet hjørnepunkt i bildekoordinater til en av QR-kodene i bildet: (300,550) piksler
- Det samme hjørnepunktet i romkoordinater fra QR-kodens streng: (50,100) mm
- Bildets senter i bildekoordinat, høyde og bredde delt på to: (640,512) piksler

Det første som blir gjort er å kompensere for rotasjonen på -10°. Dette blir gjort som forklart i teorien om geometri i bildet kap. 2.3. Det som i prinsippet skjer er at pikslene i bildet blir rotert +10° rundt senter av bildet.
Verdiene blir satt inn i ligning (8) fra kap. 2.3:

$$\begin{bmatrix} r_x' \\ r_y' \end{bmatrix} = \begin{bmatrix} \cos(10) & \sin(10) \\ -\sin(10) & \cos(10) \end{bmatrix} \begin{bmatrix} 300-640 \\ 550-512 \end{bmatrix} = \begin{bmatrix} -328 \\ 96 \end{bmatrix}$$

Koordinatene til hjørnepunktet er nå oppgitt i forhold til senter av bildet, og ved å legge til bildets senter, slik som ligning (11) fra kap. 2.3, blir punktet referert til bildets koordinatsystem igjen:

$$\begin{bmatrix} -328 + 640 \\ 96 + 512 \end{bmatrix} = \begin{bmatrix} 312 \\ 608 \end{bmatrix}$$

Resultatet av denne operasjonen er at romkoordinatsystemet og bildekoordinatsystemet blir parallelle.

Neste steg i prosessen er å finne en vektor, v, som går fra hjørnepunktet til senter av bildet i piksler. Den finnes enkelt ved å trekke hjørnepunktet fra senterpunktet:

$$v = \begin{bmatrix} 640 - 312 \\ 512 - 608 \end{bmatrix} = \begin{bmatrix} 328 \\ -96 \end{bmatrix} piksler$$

Nå er en vektor mellom hjørnepunktet og senter kjent i piksler. Størrelsen på en piksel i millimeter er også kjent. Vektoren kan da multipliseres med pikselstørrelsen slik at den blir oppgitt i millimeter, y-leddet i vektoren blir også multiplisert med (-1), dette er fordi koordinatsystemene har motsatt retning på y-aksene:

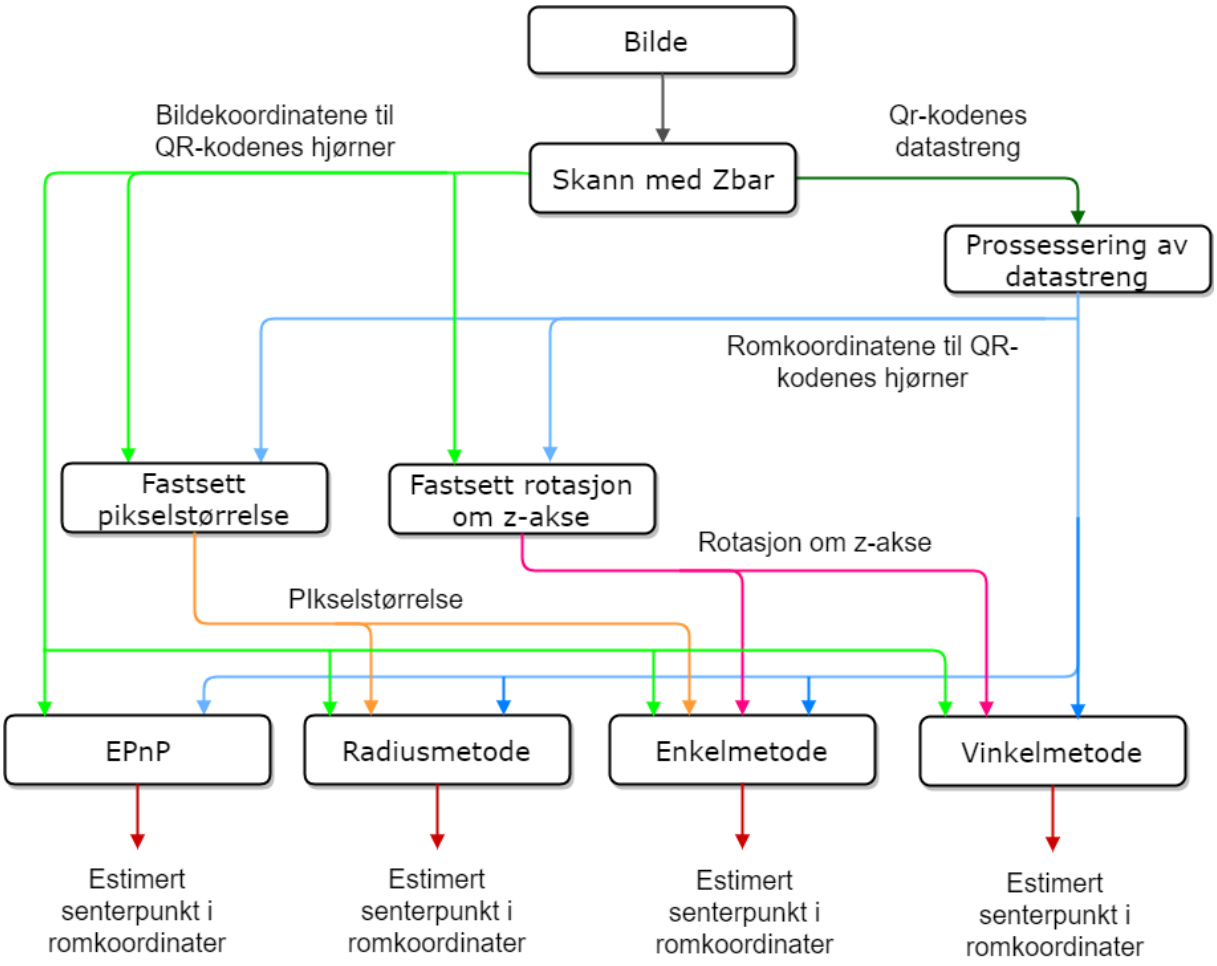
$$v \cdot 0{,}1mm = \begin{bmatrix} 32{,}8 \\ 9{,}6 \end{bmatrix} mm$$

Hjørnepunktet i romkoordinater er kjent som (50,100) mm. Ved å legge vektoren i millimeter til dette punktet, blir senterpunktets romkoordinat funnet:

$$\begin{bmatrix} 50 \\ 100 \end{bmatrix} mm + \begin{bmatrix} 32{,}8 \\ 9{,}6 \end{bmatrix} mm = \begin{bmatrix} 82{,}8 \\ 109{,}6 \end{bmatrix} mm$$

I Eksempel 1 er senterpunktet funnet ut fra ett hjørnepunkt i en singel QR-kode. I en ideell verden vil dette fungere utmerket, alle QR-koder vil finne den samme verdien for senterpunktet til bildet. I realiteten vil det oppstå forskjell i beregningene fra ulike QR-koder og ulike hjørnepunkter. Disse forskjellene eller feilene kan komme fra linseforvrenging, feil i målinger av pikselstørrelsen og feil i målinger av rotasjon om z-aksen. Det kan oppstå feil og unøyaktigheter i ZBar sin estimering av bildekoordinatene til QR-koden. Er det litt perspektiv på kameraet, vil det oppstå små feil fra perspektivforvrenging. Dette og gjerne flere ukjente parametere gjør at det å gå enkleste vei fra det nærmeste hjørnet ikke nødvendigvis vil gi det mest presise estimatet av posisjonen. De fleste algoritmene som er utviklet i prosjektet er derfor funksjoner av flere QR-koder.

Videre i dette kapittelet er algoritmene som er konstruert i prosjektet presentert. Det er laget flere uavhengige algoritmer for å finne rotasjon rundt z-aksen, og det er flere uavhengige algoritmer for å finne pikselstørrelsen. Det er til slutt flere uavhengige algoritmer for å estimere posisjonen til senter av bildet basert på disse to parameterne. Algoritmene gjør bruk av flere QR-koder og mer enn ett av hjørnene på ulike måter. Figur 12 kan være nyttig for å få en oversikt over hvilke typer algoritmer som finnes i prosjektet og hvilke inn- og utgangsparametere de har. I kap. 4 er algoritmene testet mot hverandre.



Figur 12. Oversikt over prosessen i systemet. Her er det tydelig hvilke algoritmer som trenger hvilke parametere som inngang, og hva som er resultatet av algoritmene. Rektanglene indikerer algoritmer eller metoder, og pilene viser inn- og utgangsparametere.

3.1 Algoritmer for å fastsette kameraets rotasjon om z-aksen

Kameraets rotasjon rundt z-aksen til romkoordinatene, eller vinkelen mellom romkoordinatsystemet og bildekoordinatene er funnet på flere måter. Det er laget algoritmer som baserer seg både på single QR-koder og som funksjon av flere.

Rotasjon defineres som orienteringen til bildet addert med vinkelen til bildet. Vinkelen går fra -90° til +90°. Orienteringen er den grove rotasjonen til QR-kodene, 0° eller 180°.

Orientering

Fra resultatet av å skanne et bilde med ZBar finnes hjørnekoordinatene til hver QR-kode sortert i slik rekkefølge som tallene i Figur 13 viser. Denne rekkefølgen følger orienteringen til QR-koden slik at om QR-koden er rotert 180° så vil hjørne 0 være nederst til høyre. For å finne ut om QR-koden står riktig vei eller er rotert med 180°, er det laget en algoritme som ser på hvilket hjørne som har den minste y-koordinaten, med andre ord hvilket hjørne som er høyest oppe i bildet. I eksempelet i Figur 13 står QR-koden med 0° orientering. Denne orienteringen vil være lik for alle QR-koder i bildet. Oppbyggingen til denne algoritmen er vist med pseudokode i Figur 14.

Figur 13. Dette er et eksempel på en skannet QR-kode hvor bildekoordinatene som hentes ut fra ZBar er tegnet inn. Her har hjørne 0 og 3 de minste y-koordinatene. Det betyr at QR-koden står med 0° orientering. Om QR-koden hadde stått opp ned, så ville hjørne 0 og 3 være nederst i bildet og dermed ha større y-koordinat enn hjørne 1 og 2.

```
input: corners[4], //QR-kodens fire hjønerkoordianter
                   //i bildet
Run{
        minY = index_of_minvalue(corners.y);

        switch (minY)

        {
        case 0: orient =0; break;
        case 1: orient = 180; break;
        case 2: orient = 180; break;
        case 3: orient = 0; break;

        }
        return orient;  //Orientering
}
```

Figur 14. Pseudokode som viser hvordan orienteringen til QR-kodene blir funnet.

3.1.1 Singel BoundingBox

Dette er en algoritme som gjør bruk av en metode fra OpenCV, minAreaRect for å finne vinklen til en QR-kode. MinAreaRect tilpasser et rektangel rundt de punktene den får som inngangsparameter. Så mange punkter som ønskelig kan settes som inngangsparameter, og metoden returnerer et objekt som inneholder størrelsen og vinkelen til rektangelet.

Single Boundingbox bruker hjørnepunktene i QR-koden som inngangsparameter i metoden minAreaRect. Vinkelen på rektangelet blir hentet direkte fra det returnerte objektet. I Figur 13 er dette rektanglet tegnet i blått. Relevant pseudokode er skrevet i Figur 15. Her er parameteren «corners» de fire hjørnene til QR-koden. I kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getRot_singleBoundingBox`.

```
input: corners  // Samling med alle
                // hjørnepunktene til
                // den eller de QR-kodene
                // som skal brukes
Run {
        // Metode som kjøres fra OpenCV
        OpenCV_object =minAreaRect(corners);

        return OpenCV_object.getAngle();

}
```

Figur 15. Pseudokode som viser hvordan vinklelen blir funnet ved hjelp av boundingbox-algortimene. Koden er relevant for kap. 3.1.1 og 3.1.6.

```
input: corners[4], //QR-kodens fire
                   //hjønekoordianter
                   //i bildet
Run{
        //Linjer mellom hjønepunkt
        line1 = conres[0] - conres[1];
        line2 = conres[1] - conres[2];
        line3 = conres[2] - conres[3];
        line4 = conres[3] - conres[0];

        //Vinkler fra linjer
        angle1 = arctan(line1.y / line1.x);
        angle2 = arctan(line2.y / line2.x);
        angle3 = arctan(line3.y / line3.x);
        angle4 = arctan(line4.y / line4.x);

        if (line == vertical) angle = angle-90;

        return mean(angle 1-4);

}
```

Figur 16. Pseudokode som viser hvordan vinkelen blir funnet ved hjelp av Snitt av linjer. Koden er relevant for kap. 3.1.2.

3.1.2 Snitt av linjer for rotasjon

Denne algoritmen regner ut vinkelen til de fire ytterlinjene til en QR-kode og tar et gjennomsnitt av dem.

Linjene mellom to punkter blir sett på som en vektor. Vektoren er sluttpunktet minus startpunktet, med andre ord hjørne 0 minus hjørne 1 for første linje. Vinkelen blir funnet ved å ta arcustangens til forholdet mellom x- og y-komponenten til vektoren slik som vist i kap. 2.3.

Dette blir gjort for hver av de fire ytterlinjene. For de loddrette linjene trekkes det fra 90°, deretter blir gjennomsnittet av de fire vinklene funnet.

Relevant pseudokode er skrevet i Figur 16, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getRot_meanLines`.

```
input:  allQR_img     // Vektor med alle QR-koder, for hver QR-kode brukes hjørne 1 i bildet
        allQR_room    // Vektor med alle QR-koder, for hver QR-kode brukes hjørne 1 i rommet
        bool vertical // Valg om linjen skal være vertikal eller horisontal

Run{
        if (vertical) {
                sorted_QR = sortY(allQR_room);  //Sorter alle QR-koder med lik Y verdi i grupper
                maxCount = max(sorted_QR);      //Finn den gruppen med flest QR-koder
        }
        else {
                sorted_QR = sortX(allQR_room);  //Sorter alle QR-koder med lik X verdi i grupper
                maxCount = max(sorted_QR);      //Finn den gruppen med flest QR-koder
        }

        QRs_index[ ] = choose(maxCount);        //Velg ut to QR-koder som skal brukes til å
                                                //trekke en linje, valg er avhengig av algoritme

        line = allQR_img[QRs_index[1]] - allQR_img[QRs_index[2]]; //Merk: linjen finnes ikke slik
                                                                  //i algoritmen Tilpasset linje
        angle = arctan(line.y / line.x);

        if (vertical) angle = angle - 90;

        return angle;
}
```

Figur 17. Pseudokode som viser hvordan rotasjonen blir funnet ved hjelp av linjer. Koden er relevant for kap. 3.1.3, 3.1.4 og 3.1.5.

3.1.3 Lang linje for rotasjon

Denne algoritmen finner den rette linjen mellom to QR-koder som ligger på linje i QR-matrisen. Algoritmen kan brukes på både vertikale og horisontale linjer. For å finne hvilke to QR-koder som skal brukes, er det satt noen kriterier:

- Det skal plukkes QR-koder fra den linjen hvor det er detektert flest QR-koder
- Hvis to linjer har like antall QR-koder, benyttes den linjen som ligger nærmest senter av bildet
- De to QR-kodene som ligger lengst fra hverandre på denne linjen skal brukes

Informasjon om hvilke QR-koder som er på linje kalkuleres fra QR-kodene sine romkoordinater fra datastrengen. Vinkelen på linjen blir funnet med arcustangens som i kap. 2.3.

Relevant pseudokode finnes i Figur 17, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getRot_longLine`.

3.1.4 Kort linje for rotasjon

Denne algoritmen finner den rette linjen mellom to QR-koder som ligger på linje i QR-matrisen og er nærmest mulig senter av bildet. Algoritmen kan brukes på både vertikale og horisontale linjer. For å finne hvilke to QR-koder som skal brukes, er det satt noen kriterier:

- Det skal plukkes QR-koder fra den linjen hvor det er detektert flest QR-koder
- Hvis to linjer har like antall QR-koder, benyttes den linjen som ligger nærmest senter av bildet
- De to QR-kodene, på den valgte linjen, som ligger nærmest senter av bildet skal brukes

Informasjon om hvilke QR-koder som er på linje kalkuleres fra QR-kodene sine romkoordinater fra datastrengen. Vinkelen på linjen blir funnet med arcustangens som i kap. 2.3.

Relevant pseudokode finnes i Figur 17, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getRot_shortLine`.

3.1.5 Tilpasset linje for rotasjon

Dette er en algoritme som finner den rette linjen mellom alle QR-koder som er på linje og finner vinkelen til denne linjen. Algoritmen kan brukes på både vertikale og horisontale linjer. En metode fra OpenCV, FitLine, blir benyttet for å tilpasse en linje mellom en rekke punkter. Linjen tilpasses slik at den får minst mulig sammenlagt kvadrert avstand til punktene. De to øverste hjørnene i hver QR-kode blir brukt som punkter. For å finne hvilke QR-koder som skal brukes, er det satt noen kriterier:

- Det skal plukkes QR-koder fra den linjen hvor det er detektert flest QR-koder
- Hvis to linjer har like antall QR-koder, benyttes den linjen som ligger nærmest senter av bildet

Informasjon om hvilke QR-koder som er på linje kalkuleres fra QR-kodene sine romkoordinater fra datastrengen. Vinkelen på linjen blir funnet med arcustangens som i kap. 2.3.

Denne algoritmen bruker metoden FitLine fra OpenCv til å lage en linje, og den bruker to hjørnekoordinater fra hver QR-kode. Utover dette er pseudokoden i Figur 17 relevant. I kildekoden som ligger i vedlegg, kap. 7, er denne algoritmen implementert som `ScanResults::getRot_fitLine`.

3.1.6 Multi BoundingBox

Dette er en algoritme som gjør bruk av metoden minAreaRect fra OpenCV på samme måte som Singel BoundingBox i kap. 3.1.1. Metoden tilpasser et rektangel rundt de punktene den får som inngangsparameter. Så mange punkter som ønskelig kan settes som inngangsparameter. Metoden returnerer et objekt som inneholder størrelsen og vinkelen til rektangelet.

Algoritmen Multi Boundingbox finner vinkelen fra alle QR-kodene i bildet ved å bruke alle hjørnepunktene til alle QR-koden som inngangsparameter i metoden minAreaRect. Vinkelen på rektangelet blir hentet direkte fra objektet. Relevant pseudokode er skrevet i Figur 15. Her er parameteren «corners» alle hjørnene til alle QR-koder i bildet. I kildekoden som ligger i vedlegg, kap. 7, er denne algoritmen implementert som `ScanResults::getRot_multiBoundingBox`.

3.2 Algoritmer for å fastsette pikselstørrelsen i romkoordinater

Formålet med algoritmene som er beskrevet i dette kapittelet, er å finne hvor stor avstand en piksel i bildet representerer i romkoordinatsystemet. Det er laget algoritmer som baserer seg både på single QR-koder og som funksjon av flere.

Alle algoritmene er basert på å finne en linje mellom to punkter. Algoritmene finner avstanden mellom to punkter i bildet oppgitt i piksler. Den finner også avstanden i QR-matrisen oppgitt i millimeter. Pikselstørrelsen er da avstand i millimeter delt på avstand i piksel. Heretter nevens disse verdiene som mm-verdi og pikselverdi.

3.2.1 Snitt av linjer for pikselstørrelse

QR-kodene er kvadratiske og dimensjonen i millimeter er et av parameteren som blir lest fra QR-koden. ZBar finner punktet til hver av de fire hjørnene i QR-koden oppgitt i piksler.

Denne algoritmen er delt opp i tre deler og leverer tre resultater for hver QR-kode. Den først delen bruker de fire linjene mellom de fire hjørnene som pikselverdi og den leste dimensjonen som mm-verdi. Et gjennomsnitt av pikselstørrelsen fra disse fire linjene blir returnert.

Den neste delen bruker de to diagonale linjene mellom motstående hjørner som pikselverdi. Diagonalen i mm-verdi blir funnet ved Pytagoras setning med kateter lik den leste dimensjonen. Gjennomsnitt av pikselstørrelsen fra disse to linjene returneres. Den siste delen returnerer et gjennomsnitt av pikselstørrelsen til alle de seks linjene fra de to foregående delene.

Relevant pseudokode er skrevet i Figur 18, og i kildekoden som ligger i vedlegg, kap. 7, er denne algoritmen implementert som `ScanResults::getPixelSize_meanLines`.

3.2.2 Lang linje for pikselstørrelse

Denne algoritmen finner, på samme måte som Lang linje for rotasjon i kap.3.1.3, den rette linjen mellom to QR-koder som ligger på linje i QR-matrisen, og som er lengst mulig fra hverandre. For å finne hvilke to QR-koder som skal brukes, er det satt noen kriterier:

- Det skal plukkes QR-koder fra den linjen hvor det er detektert flest QR-koder
- Hvis to linjer har like antall QR-koder, benyttes den linjen som ligger nærmest senter av bildet
- De to QR-kodene som ligger lengst fra hverandre på denne linjen skal brukes

Informasjon om hvilke QR-koder som er på linje kalkuleres fra QR-kodene sine koordinater i QR-matrisen. De blir lest fra datastrengen. Mm-verdien av linjen blir funnet som avstanden mellom disse punktene, og pikselverdien av linjen blir funnet som lengden av vektoren mellom de samme hjørnene i bildet.

Relevant pseudokode finnes i Figur 19, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getPixelSize_longLine`.

```
input:  dim             //QR-kodens fysiske størrelse i rommet
        corners[4]      //QR-kodens fire hjønekoordianter i bildet
Run{
        //Linjer mellom hjønepunkt
        line1 = conres[0] - conres[1];
        line2 = conres[1] - conres[2];
        line3 = conres[2] - conres[3];
        line4 = conres[3] - conres[0];
        line5 = conres[0] - conres[2];
        line6 = conres[1] - conres[3];

        //Linjenes lengde
        length1 = sqrt(line1.x² + line1.y² );
                .
                .
        length6 = sqrt(line6.x² + line6.y² );

        //Pikselstørrelse for ytterlinjer
        pixsize1 = length1 / dim;
                .
        pixsize4 = length4 / dim;

        //Pikselstørrelse for diagonal
        pixsize5 = length5 / sqrt(2*dim²);
        pixsize6 = length6 / sqrt(2*dim²);

        return 1 mean(pixsize 1-4);
        return 2 mean(pixsize 5-6);
        return 3 mean(pixsize 1-6);
}
```

Figur 18. Pseudokode som viser hvordan pikselstørrelsen blir funnet ved hjelp av Snitt av linjer. Koden er relevant for kap. 3.2.1.

```
input:  allQR_img     // Vektor med alle QR-koder, for hver QR-kode brukes hjørne 1 i bildet
        allQR_room    // Vektor med alle QR-koder, for hver QR-kode brukes hjørne 1 i rommet
        bool vertical // Valg om linjen skal være vertikal eller horisontal
                      // Merk at ved bruka algoritmen lengst mulig linje velges ikke retning,
                      // if og else setningene utgår
Run{
        if (vertical) {
                sorted_QR = sortY(allQR_room);  //Sorter alle QR-koder med lik Y verdi i grupper
                maxCount = max(sorted_QR);      //Finn den gruppen med flest QR-koder
        }
        else {
                sorted_QR = sortX(allQR_room);  //Sorter alle QR-koder med lik X verdi i grupper
                maxCount = max(sorted_QR);      //Finn den gruppen med flest QR-koder
        }

        QRs_index[2] = choose(maxCount);        //Velg ut to QR-koder som skal brukes til å
                                                //trekke en linje, valg er avhengig av algoritme

        line_img = allQR_img[QRs_index[1]] - allQR_img[QRs_index[2]];
        line_room = allQR_room[QRs_index[1]] - allQR_room[QRs_index[2]];

        length_img = sqrt(line_img.x² + line_img.y²);    //Lengden mellom QR-kodene i bildet
        length_room = sqrt(line_room.x² + line_room.y²); //Lengden mellom QR-kodene i rommet

        return length_room/length_img;
}
```

Figur 19. Pseudokode som viser hvordan pikselstørrelsen blir funnet ved hjelp av en linje. Koden er relevant for kap. 3.2.2, 3.2.3 og 3.2.4.

3.2.3 Kort linje for pikselstørrelse

Denne algoritmen finner, på samme måte som Kort linje for rotasjon i kap.3.1.4, den rette linjen mellom to QR-koder som ligger på linje i QR-matrisen og som er nærmest mulig senter av bildet. For å finne hvilke to QR-koder som skal brukes er det satt noen kriterier:

- Det skal plukkes QR-koder fra den linjen hvor det er detektert flest QR-koder
- Hvis to linjer har like antall QR-koder, benyttes den linjen som ligger nærmest senter av bildet
- De to QR-kodene på denne linjen, som ligger nærmest senter av bildet skal brukes

Informasjon om hvilke QR-koder som er på linje kalkuleres fra QR-kodene sine koordinater i QR-matrisen. De blir lest fra datastrengen. Mm-verdien av linjen blir funnet som avstanden mellom disse punktene. Pikselverdien av linjen blir funnet som lengden av vektoren mellom de samme hjørnene i bildet.

Relevant pseudokode finnes i Figur 19, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getPixelSize_shortLine`.

3.2.4 Lengst mulig linje for pikselstørrelse

Denne algoritmen finner linjen mellom de to QR-kodene som har størst avstand til hverandre. Dette kalkuleres med en iterativ fremgangsmåte. Hver QR-kodes avstand i QR-matrisen til alle andre QR-koder kalkuleres, og de to med lengst avstand til hverandre blir valgt. Pikselstørrelsen blir så kalkulert på lik måte som i kap.3.2.2 og 3.2.3.

Relevant pseudokode finnes i Figur 19, men her velges ikke noen retning som inngangsparameter. If og else setningene i figuren går dermed ut for denne algoritmen. I kildekoden som ligger i vedlegg, kap. 7, er denne algoritmen implementert som `ScanResults::getPixelSize_longestPossibleLine`.

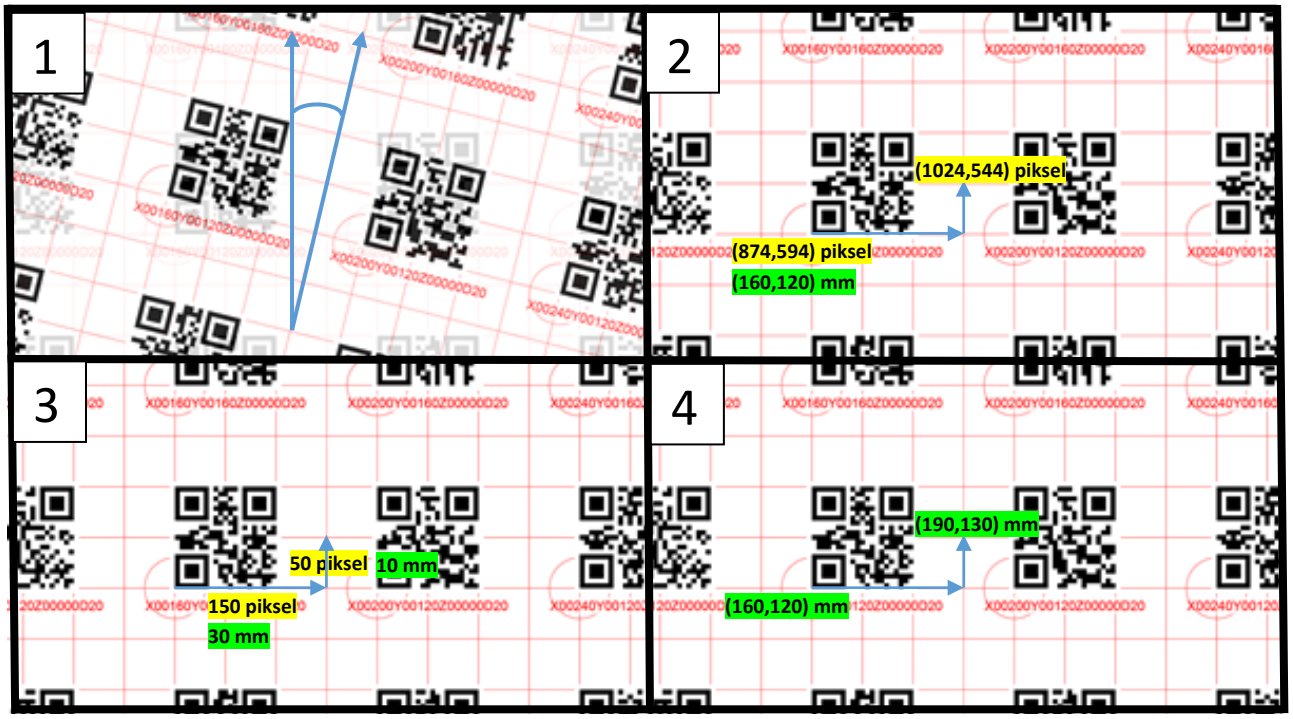
3.3 Algoritmer for å estimere senterpunktet til bildet i romkoordinater

Som oppgaveteksten fra kap. 1.2 sier, er målet med oppgaven å finne gode metoder for å lokalisere bildets senter i forhold til et fast koordinatsystem i rommet. Som bakgrunn for dette, trengs det noen parametere. De blir brukt i større eller mindre grad i de ulike algoritmene for å estimere posisjonen. Disse parameterne er:

- QR-koden sine hjørnekoordinater i forhold til bildets origo. De hentes direkte fra resultatet av QR-scann med ZBar. Se kap. 2.1. Videre blir disse punktene referert til som bildepunkter, eller bildekoordinater.
- QR-koden sine hjørnekoordinater i forhold til koordinatsystemet i rommet. De hentes ut fra datastrengen som er lagret i QR-koden. Videre blir disse punktene referert til som rompunkter, eller romkoordinater.
- Rotasjonen til bildet rundt Z-aksen blir funnet med algoritmene fra kap.3.1.
- Pikselstørrelsen blir funnet med algoritmene fra kap. 3.2.

Disse fire parameterne regnes som kjent før senterpunktet lokaliseres. Videre kommer en forklaring rundt de tre algoritmene som er utviklet i prosjektet for å lokalisere bildets senter i romkoordinater.

3.3.1 Enkelmetode



Figur 20. Talleksempel på Enkelmetode. I rute 1 er rotasjonen rundt z-aksen indikert med bildekoordinatsystemet i framgrunnen og romkoordinatsystemet som svak bakgrunn. I overgangen til rute 2 har det blitt korrigert for rotasjonen. Her er bildekoordinatene til ett av hjørnene til QR-koden og senteret av bildet tegnet inn i tillegg til de kjente romkoordinatene. I rute 3 er vektoren fra hjørnet til senterpunkt tegnet inn. Den målte pikselverdien er markert gult og den beregnede mm-verdien i grønt. I dette eksempelet er pikselstørrelse 0,2 mm brukt til å beregne vektoren i mm. Rute 4 viser hjørnepunktet og det beregnede senterpunktet i romkoordinater. Y-aksen i romkoordinatsystemet går i motsatt retning av bildekoordinatsystemet. Det rettes ved å bytte fortegn på y-delen av retningsvektoren.

Enkelmetode er en algoritme for å finne senterpunktet til bildet i romkoordinater. Den tar utgangspunkt i bildepunktet til hjørne av en QR-kode. Det første som gjøres er å kompensere for rotasjonen rundt z-aksen. Det er gjort som i kap. 2.3. Bildepunktet roteres rundt senter i motsatt retning av den kjente rotasjonen. Dette resulterer i at bildepunktet får rotasjon rundt z-aksen lik 0, og da har bildekoordinatsystemet den samme orienteringen som romkoordinatsystemet, se Figur 20, rute 1 til 2.

Deretter blir avstandsvektoren i piksler mellom bildets senter og QR-koden funnet ved å trekke bildepunktet til senter fra QR-koden sitt bildepunkt, Figur 20, rute 2 til 3.

Ved å multiplisere avstandsvektoren med den kjente pikselstørrelsen, blir vektoren transformert fra bildekoordinater til romkoordinater. Vektoren får benevnelsen millimeter. Se Figur 20, rute 3, her er pikselstørrelse 0,2 mm brukt som eksempel. Y-komponenten til vektoren bytter fortegn for å kompensere for at bildekoordinatsystemet og romkoordinatsystemet har motsatt retning på y-aksene.

Romkoordinatene til bildets senter blir nå funnet ved å addere hjørnepunktet til QR-koden i romkoordinater med den transformerte avstandsvektoren, se Figur 20, rute 4.

Denne prosedyren gjøres på hvert av hjørnene til hver av QR-kodene, og et gjennomsnitt av de estimerte koordinatene returneres.

Relevant pseudokode til Enkelmetode finnes i Figur 22, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getCp_strigthCalc`.

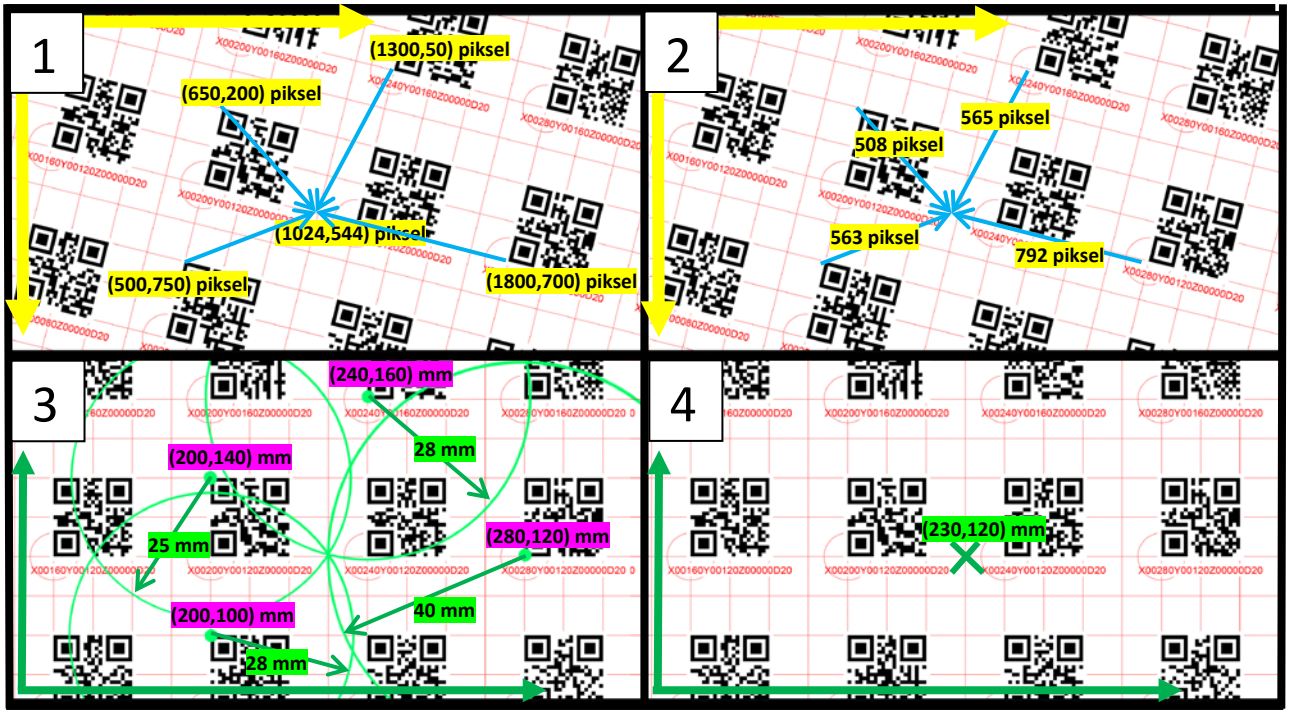
```
Input:  pixelSize       // Pikselstørrelsen
        rotation        // Rotasjon rundt z-aksen
        corner_img      // Tabell med alle hjørnepunkter i bildet fra alle QR-koder
        corner_room     // Tabell med alle hjørnepunkter i rommet fra alle QR-koder
        cp_img          // Senterpunktet i bildekoordinater


        Run{
        for i++{
                //Roter bildepunkter med ligninger fra kap. 2.3
                rot_corn_img = compensate_for_rot(corners_img[i],rotation);

                //Vektor fra hjørnepunkt til senterpunkt i bildet
                vector_img = cp_img - rot_corn_img;

                //Transformerer vektor til romkoordinat
                vector_room = vector_img*pixelSize;

                //Kompenserer for motsatt retning på y-akse i koodrinatsystemene
                vector_room.y = (-1)*vector_room.y;

                //Finner senterpunktet ved å legge vektoren til hjørnepunktet i romkoordinat
                cp_room[i] = corner_room[i] + vector_room;
        }
        return mean(cp_room);
}
```

Figur 22. Pseudokode som viser hvordan Enkelmetode fungerer.

```
Input:  pixelSize       // Pikslstørrelsen
        corner_img      // Tabell med alle hjørnepunkter i bildet fra alle QR-koder
        corner_room     // Tabell med alle hjørnepunkter i rommet fra alle QR-koder
        cp_img          // Senterpunktet i bildekoordinater


        Run{
        for i++{

                //Vektor fra hjørnepunkt til senterpunkt i bildet
                vector_img = cp_img - corn_img[i];

                //Lengden av vektoren i bildet
                radi_img = sqrt(vector_img.x^2 + vector_img.y^2);

                //Lengden av vektoren i rommet
                radi_room = radi_img*pixelSize;

                //Sett verdier i passende matriser for least square løsning, se kap. 2.4
                [A[i], b[i]] = put_to_matrix(radi_room, corner_room);

        }
        //Løs ligningsett med opencv
        solution = OpenCV_solve_with_SVD(A,b);

        return solution;
}
```

Figur 21. Pseudokode som viser hvordan Radiusmetode fungerer.

3.3.2 Radiusmetode



Figur 23. Talleksempel på Radiusmetode. Rute 1 viser noen av punktene som hentes fra bildet. Verdier fra bildet er markert gult. Rute 2 viser neste steg der lengden av vektorene blir funnet. Videre til rute 3 blir lengden multiplisert med pikselstørrelsen. I dette eksempelet er pikselstørrelsen satt til 0,05 mm. I rute 3 og 4 brukes romkoordinatsystemet. Rotasjonen i bildet og retningen på vektorene blir irrelevant. Senterpunktet er det punktet som skjærer sirkler med radius lik lengden av vektorene i millimeter.

Dette er en algoritme for å finne senterpunktet til bildet i romkoordinater. Den er utviklet basert på teorien om trilaterasjon i kap. 2.4, og er inspirert av hvordan satellittnavigasjonssystemer som GPS fungerer [27].

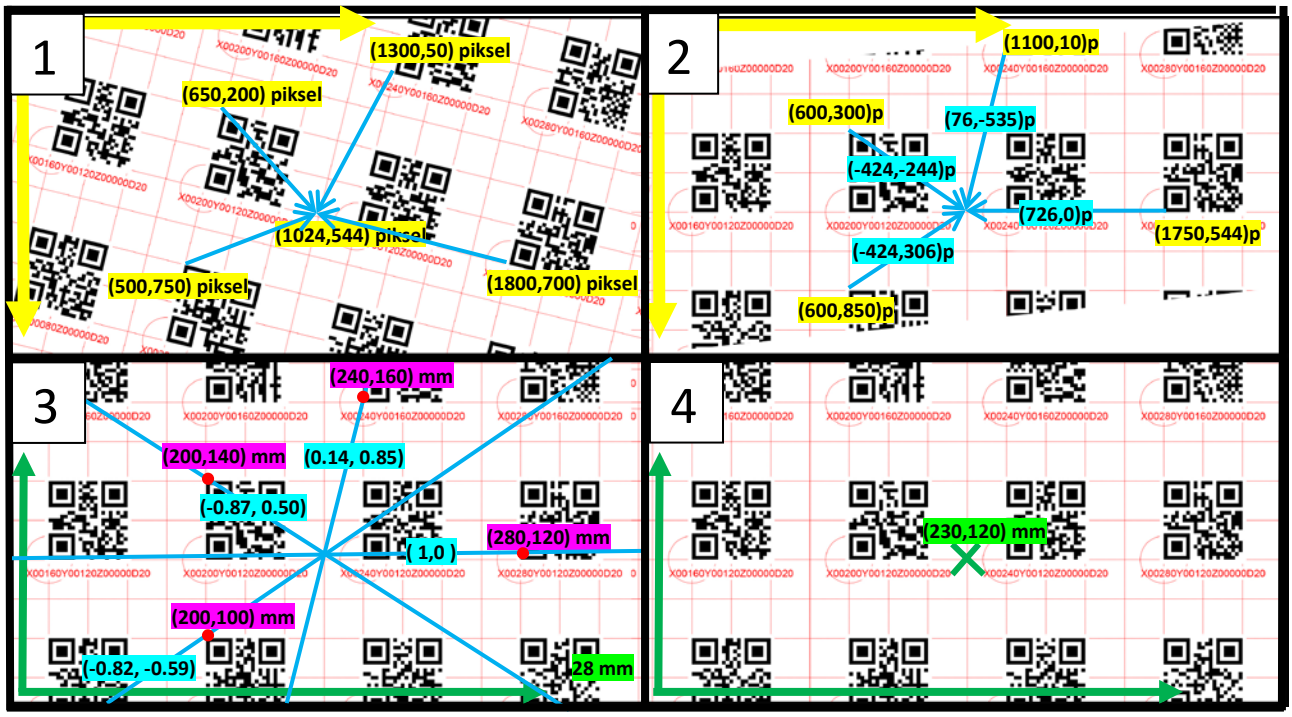
Lengden av vektoren mellom hjørnet i QR-koden og bildets senter i piksler blir funnet fra bildet. Se talleksempel i Figur 23, rute 1 og 2. Dette er et eksempel, derfor stemmer tallene ikke nødvendigvis 100 prosent. Det er også bare noen av hjørnepunktene som er tegnet inn, men i realiteten brukes alle hjørnepunktene.

Avstanden i piksler blir multiplisert med den kjente pikselstørrelsen, og da er lengden tilpasset romkoordinatsystemet. Det foreligger ikke noen informasjon om retningen til denne vektoren lenger, og den sees derfor på som en radius slik som Figur 23, rute 3 viser. Det blir laget sirkler med senterpunkt i rompunktet til hjørnet og med radius lik denne kalkulerte lengden. Dette gjøres for alle hjørnene i alle QR-kodene i bildet. Senterpunktet blir estimert der sirkler med disse radiene krysses. Dette krysningspunktet blir funnet med minste kvadraters metode slik som forklart i kap.2.4.

Her er det verdt å legge merke til at ingen informasjon om vinkel, eller rotasjon i bildet er benyttet for å komme frem til senterpunktet. Det trengs dermed en parameter mindre enn for Enkelmetode i kap 3.3.1. På den andre siden trengs det her minimum tre punkter, men i denne sammenhengen vil det ikke være ett problem siden en QR-kode har fire kjente hjørner.

Relevant pseudokode til Radiusmetode er presentert i Figur 21, og i kildekoden som ligger i vedlegg kap. 7 er denne algoritmen implementert som `ScanResults::getCp_circleIntersection`.

3.3.3 Vinkelmetode



Figur 24. Talleksempel på Vinkelmetode. Rute 1 viser de lokaliserte bildekoordinatene markert med gult. Videre til rute 2 er det kompensert for rotasjon rundt z-aksen. Vektoren mellom hjørnepunkt og senter er markert med blått. I overgangen til rute 3, er retningsvektoren normalisert til å ha lengde lik 1. Nå brukes romkoordinatsystemet. Y-verdiene har byttet fortegn som kompensasjon for at koordinatsystemene har motsett retning på y-aksene. Vektorene strekkes fra minus uendelig til pluss uendelig siden lengden er ukjent. Senterpunktet i rute 4 blir funnet ved linjenes krysningspunkt.

Dette er en algoritme for å finne senterpunktet til bildet i romkoordinater. Den er utviklet basert på teorien om triangulering i kap. 2.5, og er inspirert av Angle Of Arival(AoA) trackingsystemer [28].

Det første som blir gjort er å kompensere for rotasjon om z-aksen på lik måte som i Enkelmetode, kap. 3.3.1. Se overgang fra rute 1 til 2 i Figur 24. Etter rotasjonen har punktene fått nye pikselverdier. Nå har bildekoordinatsystemet (rute 2 i Figur 24) og romkoordinatsystemet (rute 3 og 4 i Figur 24) den samme orienteringen.

Vektoren mellom bildepunktet og senterpunket blir brukt som retningsvektor for en linje. Den normaliseres slik at lengden blir 1. Dette gjøres ved å dividere x- og y-komponentene på lengden av vektoren. Retningsvektoren er markert med blått i Figur 24, og i rute 3 er den normalisert. Her er y-komponentens fortegn byttet for å kompensere for at y-aksen går i motsatt retning i bilde- og romkoordinatsystemene. Dette blir gjort for hvert hjørnepunkt i hver QR-kode i bildet slik at det er en retningsvektor til hvert hjørnepunkt.

Det blir laget linjer som går gjennom de kjente romkoordinatene, og som har retning slik som de normaliserte retningsvektorene. Her er ingen informasjon om linjens lengde kjent, og linjene lages derfor teoretisk sett fra pluss uendelig til minus uendelig.

Ved å bruke minste kvadraters metode blir krysningspunktet til linjene funnet slik som forklart i kap. 2.5. Dette krysningspunktet er bildets senter i romkoordinater.

Siden det ikke brukes noen informasjon om lengden av vektorene i denne algoritmen, trengs det ingen informasjon om pikselstørrelsen. Det trengs dermed en parameter mindre enn for Enkelmetode i kap 3.3.1. På den andre siden trengs det her minimum to punkter, men i denne sammenhengen har det lite betydning siden en QR-kode har fire kjente hjørner.

Relevant pseudokode til algoritmen er presentert i Figur 25, og i kildekoden som ligger i vedlegg kap. 7, er denne algoritmen implementert som `ScanResults::getCp_linesIntersection`.

```
Input:  rotation        // Rotasjon rundt z-aksen
        corner_img      // Tabell med alle hjørnepunkter i bildet fra alle QR-koder
        corner_room     // Tabell med alle hjørnepunkter i rommet fra alle QR-koder
        cp_img          // Senterpunktet i bildekoordinater

        Run{
        for i++{
                //Roter bildepunkt med ligninger fra kap. 2.3
                rot_corn_img = compensate_for_rot(corners_img[i],rotation);

                //Vektor fra hjørnepunkt til senterpunkt
                vector_img = cp_img - rot_corn_img

                //Lengden av vektoren i bildet
                length_img = sqrt(vector_img.x^2 + vactor_img.y^2);

                //Normaliser vektor til lengde 1
                vector_norm = vector_img/length_img;

                //Sett verdier i passende matriser for least square løsning, se kap. 2.4 og 2.5
                [A[i], b[i]] = put_to_matrix(vector_norm, corner_room);

        }
        //Løs ligningsett med OpenCV
        solution = OpenCV_solve_with_SVD(A,b);

        return solution;
}
```

Figur 25. Pseudokode som viser hvordan Vinkelmetode fungerer.

4. Forsøk og resultater

Prosjektets hovedmål er å kunne gi en grunngitt anbefaling av hvilken metode som fungerer best til å lokalisere senter av et bilde. For å kvalitetssikre dette resultatet, er systemet delt opp i flere deler. Hver del blir testet og vurdert separat for å finne den beste løsningen. Dette er gjort for å begrense antall parametere som påvirker resultatene.

Det som blir testet i forsøkene, eller testene er:

- Algoritmer for å finne rotasjon om z-aksen
- Algoritmer for å finne pikselstørrelsen i romkoordinater
- Algoritmer for å lokalisere senter av bildet i romkoordinater
- Ideell størrelse og antall QR-koder i bildet
- Bilder fra objektiver med ulike brennvidder
- Påvirkning ved å redusere oppløsningen til bildet
- Påvirkning ved å endre vinkelen til kameraet i forhold til QR-matrisen
- Fastsette presisjon til systemet
- Påvirkninger ved bevegelse i kamera

Algoritmene som er testet er beskrevet tidligere, i kap. 3. I tillegg er algoritmen EPnP for posisjonsestimat testet. Denne algoritmen er beskrevet i teorikapittelet 2.7.

Rotasjon om z-aksen og pikselstørrelse er to parametere som algoritmene for å estimere posisjon trenger en eller begge av for å fungere, se kapittel 3. Testene av disse to parameterne er derfor kjørt først i kapittel 4.2 og 4.3, slik at resultatet kan brukes videre i de andre testene.

Kapittel 4.4 tar for seg de ulike algoritmene for estimat av senterposisjonen til bildet. Her er målet å finne den eller de algoritmene som fungerer best. Det er også sett på om det bør settes begrensinger for hvilke QR-koder som skal være med i estimatet. Figur 12 fra kap. 3 gir en oversikt over prosessen i systemet. Den kan brukes til å holde oversikt over hvilken rekkefølge algoritmene kjøres i og hvilke inngangsparametere som er nødvendige.

Kapittel 4.5 til 4.10 tar for seg ytre påvirkninger på presisjonen for posisjonsestimat, samt hvilken presisjon som er oppnåelig.

I flere av testene i kapittelet brukes standardavvik ved et stort antall målinger for å sammenligne algoritmer. Standardavviket sier noe om spredningen til målingene. Ved å anta at målingene er normalfordelte vil 99.7% av målingene være innenfor ± 3 standardavvik fra gjennomsnittet. Det blir lagt stor vekt på et lavt standardavvik i testene fordi en feil på gjennomsnittlige estimater, med lavt standardavvik, gjerne kan komme av et upresist referansepunkt. En slik feil kan eventuelt lett kompenseres for. Er det derimot et stort standardavvik, viser det at estimatet er upresist fra måling til måling. Det vil ikke kunne kompenseres for på samme måte og må uansett regnes som en feilmargin.

Det er flere ting som kan forårsake feil i posisjonsestimatet. Testene er gjort på en slik måte at feilkilder skal kunne fanges opp der de oppstår, og dermed kan de isoleres og elimineres. Mulige feilkilder kan være:

- Feil i estimatet av rotasjon
  Dette blir detektert i egen test for rotasjon. I tillegg er det bare 2 av 4 algoritmer for posisjonsestimat som gjør nytte av rotasjon. Et avvik vil oppstå mellom disse estimatene.

- Feil i estimatet av pikselstørrelse
  Dette blir detektert i egen test for pikselstørrelse. I tillegg er det bare 2 av 4 algoritmer for posisjonsestimat som gjør nytte av pikselstørrelse. Et avvik vil oppstå mellom disse estimatene.

- Feil i bildekoordinatene til en eller flere QR-koder
  Testene for estimat av posisjon har en innledende test hvor posisjonen estimeres for hver enkelt QR-kode. Her er det mulig å se avvik om en QR-kode skulle ha feil bildekoordinater.

- Feil i reell fasitverdi
  Standardavvik mellom estimater av posisjon fra flere hundre målinger blir brukt som sammenligningsgrunnlag. Det vil ikke la seg påvirke av en feil i fasiten.

4.1 Testoppsett

I prosjektet er det laget to ulike testoppsett hvor det første benytter seg av en samling bilder med manuelt funnet fasitverdier for rotasjon om z-akse, pikselstørrelse og senterposisjon. Det andre testoppsettet er satt opp for å kunne ta mange bilder av det samme motivet slik at gjennomsnitt og standardavvik kan brukes som resultater.

Testoppsett 1, Bilder

Oppsettet er basert på tolv naturlige bilder av en QR-matrise. På hver av de er det teoretisk mulig å detektere opp mot tretti QR-koder. I tillegg til disse tolv naturlige bildene, er det to bilder som kalles ideelle bilder. De er generert på PC med samme oppløsning som kameraet har. Dette er gjort for å unngå alle påvirkninger fra kameraet, og dermed finne ut hvilke resultater det er teoretisk mulig å oppnå før de naturlige bildene testes.

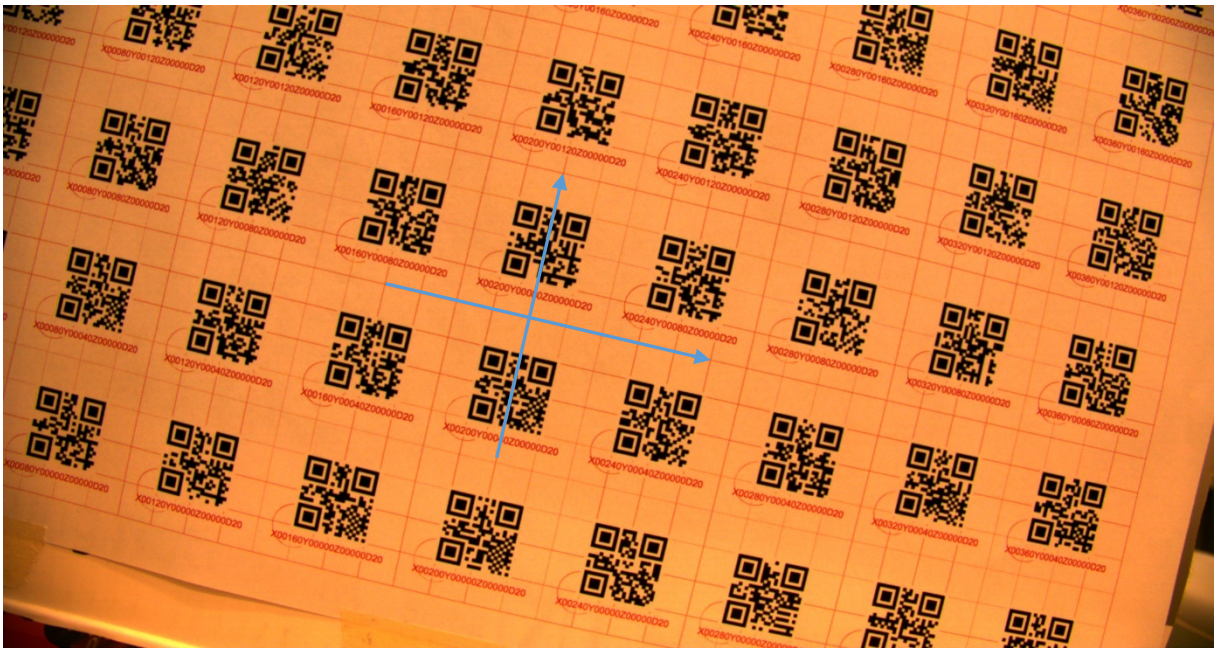
De 12 testbildene er tatt med forskjellig rotasjon om z-aksen, fra -15° til +15°. Halvparten av bildene er med linseforvrengning, og for de resterende er linseforvrengning rettet opp. Figur 26 og Figur 27 viser henholdsvis et testbilde med, og et uten forvrengning.

For å sette en fasit til rotasjonen og pikselstørrelsen er Adobe Photoshop nyttet. Vinklene til rutenettet bak QR-kodene er målt manuelt med presisjon på pikselnivå. I Figur 26 er det tegnet blå piler for å indikere de linjene som rotasjonen er målt fra. De samme linjenes lengde er målt manuelt i pikselverdi og i millimeterverdi for å gi en fasit til pikselstørrelsen. De manuelle målingene er gjort i område rundt senter av bildet for at fasiten skal ha minst mulig påvirkning fra linseforvrengning.

Bildene er tatt med et digitalt industrikamera som er beskrevet i Tabell 1. Bildene er lagret som Bitmapfiler for å unngå feil fra komprimering. De lastes inn i Visual studio og blir behandlet som OpenCV-matriser. Alle algoritmer kjøres i c++, og kildekoden kan sees i vedlegg kap. 7.

Tabell 1. Komponenter i testoppsett 1.

Kamera	IDS ui-336xcp-c Oppløsning: 2048x1088
Objektiv	Fujinon HF8XA-1 Brennvidde: 8 mm



Figur 26. Et av tolv bilder fra Testoppsett 1, uten korrigering for linseforvrengning. Bildet er rotert 13.1°. De blå pilene indikerer hvor vinkelen og pikselverdien på forhånd er målt manuelt. Legg merke til at motivet ser ut til å bule utover, dette er resultat av linseforvrengningen.

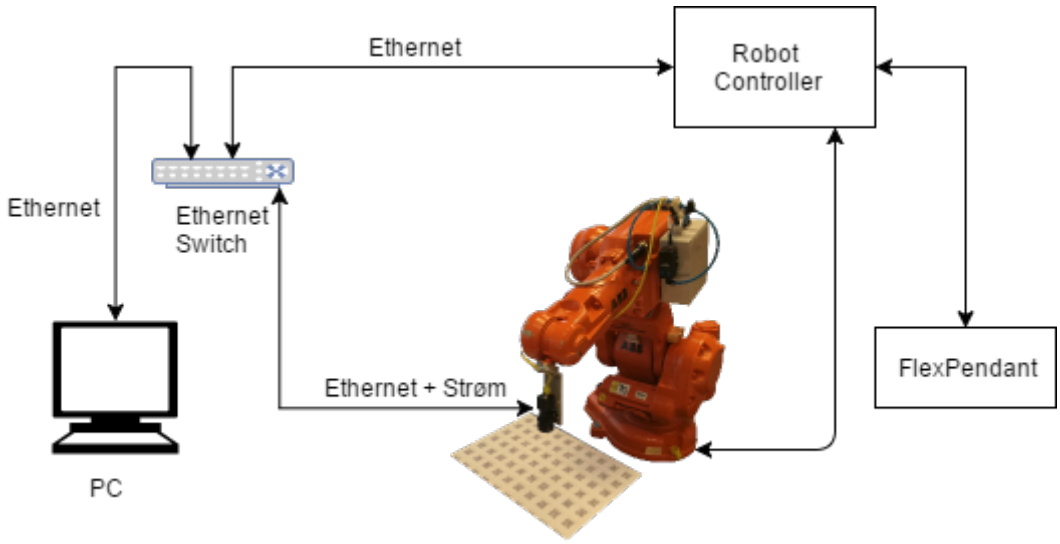
Figur 27. Dette er det samme bildet som i Figur 26, men her er det gjort korrigering for linseforvrenging. Legg merke til de svarte feltene langs bildekanten. Dette er korrigering for linseforvrengningen.

Testoppsett 2, Robot

Testoppsett 2 er laget for testene i kapittel 4.4 til 4.10. I disse testene er resultatet den estimerte posisjonen til senter av bildet. Målet med oppsettet er å ha en rigg som kan ta flere hundre bilder med eksakt det samme senterpunktet. Det gir grunnlag for å bruke gjennomsnitt og standardavvik til senterpunktene som resultater i testene. Komponentene i testoppsettet er beskrevet i Tabell 2 og Figur 28 viser en grafisk fremstilling.

Tabell 2. Komponenter i testoppsett 2.

Komponent	**Spesifikasjon**
Industrikamera	PointGray BFLY-PGE-13E4C-CS Oppløsning: 1280x1024 Framerate: 60fps
Objektiv	Brennvidde: 8 mm
Industrirobot	ABB *IRB 140*
Ethernet switch	Med Power over Ethernet
Pc	Windows 7, 64bit Intel i5, 3.1 GHz 16GB ram
Diverse	Ethernetkabler



Figur 28. Det fysiske komponentene i Testoppsett 2. All kontakt mellom PC, kamera og robot går på ethernet. Kameraet er montert i verktøyflensen til en robot, og det får både strøm og data over ethernetkabelen. Styring av roboten kan gjøres fra FlexPendanten eller fra PC. Roboten er uavhengig av kamera og bildebehandling selv om de er koplet gjennom det samme nettverk.

Industrikameraet er montert på verktøyflensen til roboten, og en QR-matrise er festet på et bord i arbeidsområdet til roboten. Roboten kan brukes til å flytte kameraet til den eksakte ønskede posisjonen og avstanden over QR-matrisen. Dette gjør at kameraet og QR-matrisen kan posisjoneres helt likt under lange tester.

Kameraet er koplet til en PC via lokalt nettverk og det er kraftforsynt av Power Over Ethernet. De kommuniserer med hverandre over protokollen GIGE Vision. Et grensesnitt fra kameraleverandøren gjør det enkelt å bruke bildene fra kameraet direkte i den koden som er laget i dette prosjektet. Se kildekoden i vedlegg kap. 7.

Styringen av roboten gjøres uavhengig av bildebehandlingen. For testene i kap. 4.4 til 4.9 er roboten styrt manuelt med joysticken på FelxPendanten. For testen i kap 4.10, hvor det blir kjørt en fast bane, er det laget et program i programmeringsspråket Rapid som lastes inn på roboten og kjøres.

4.2 Test av algoritmer for å fastsette rotasjon om z-aksen

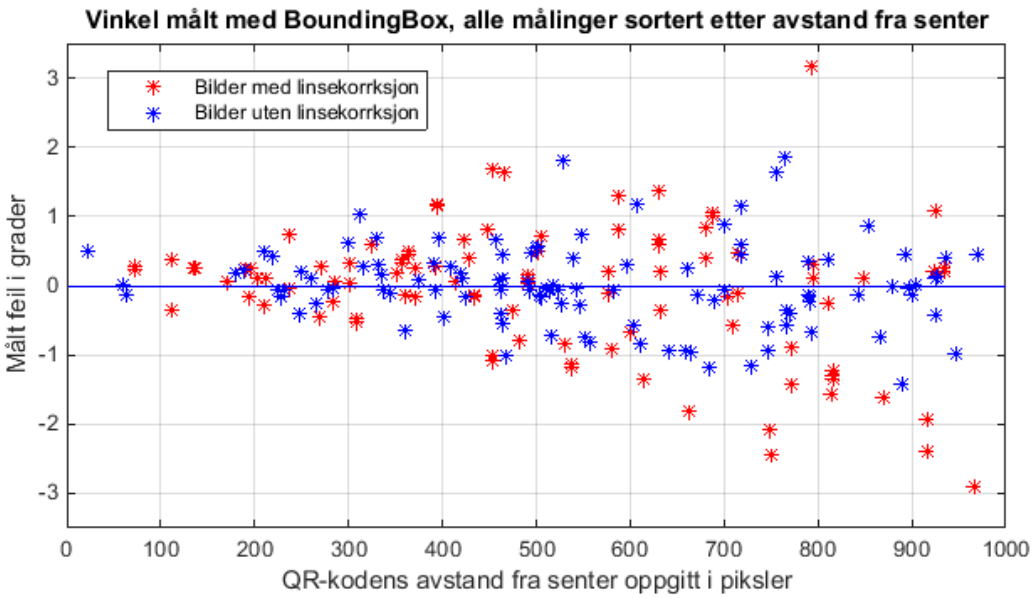
Testene av algoritmer for å finne rotasjon rundt z-aksen er delt opp i to deler. Den første ser på hver QR-kode alene. Den andre delen ser på flere QR-koder om gangen. I begge delene er Testoppsett 1 fra kap 4.1 brukt.

4.2.1 Rotasjon fra single QR-koder

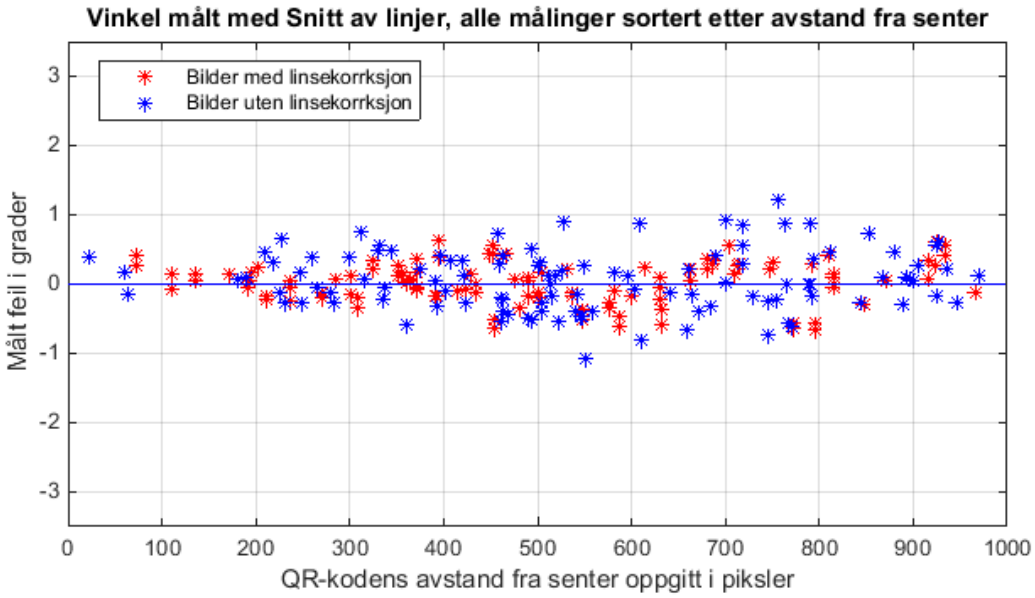
For å se om plasseringen til QR-kodene har sammenheng med målefeil, er det kjørt en test av algoritmene Singel BoundingBox og Snitt av linjer fra kap. 3.1.1 og 3.1.2. Testene viser estimat av rotasjonen for hver enkelt QR-kode i bildet.

Resultatene er presentert i Figur 29 og Figur 30. Snitt av linjer har mindre spredning i resultatene enn BoundingBox. Spredningen i resultatene øker mye for de QR-kodene som har større avstand enn 400 til 500 piksel fra senter av bildet. Dette gjelder spesielt for Singel BoundingBox.

For begge algoritmer er målefeilene spredd jevnt over og under null. Snitt av linjer har mindre spredning for de bildene med linsekorreksjon enn de uten. For BoundingBox er resultatet motsatt.



Figur 29. Målefeil sortert etter QR-kodens avstand fra senter. Dette plottet viser resultatene fra algoritmen Singel BoundingBox, for hver enkelt QR-kode.



Figur 30. Målefeil sortert etter QR-kodens avstand fra senter. Dette plottet viser resultatene fra algoritmen Snitt av linjer, for hver enkelt QR-kode.

4.2.2 Rotasjon fra flere QR-koder

For å finne de beste resultatene for rotasjonen om z-aksen, er flere algoritmer basert på mer enn en QR-kode testet mot hverandre.

Følgende algoritmer er testet:

- Lang linje horisontalt kap. 3.1.3
- Kort linje horisontalt kap. 3.1.4
- Tilpasset linje horisontalt kap. 3.1.5
- Tilpasset linje Vertikalt kap. 3.1.5
- Multi Boundingbox kap. 3.1.6

Resultatene fra testen av single QR-koder, viser målefeil som er spredd jevnt over og under null. Dette indikerer at et gjennomsnitt av flere QR-koder kan gi bra resultat, og derfor er et gjennomsnitt av resultatene fra algoritmene for single QR-koder testet sammen med algoritmene for flere QR-koder. Testen av single QR-koder viser også at QR-koder som er lenger fra senter av bildet kan gi større feil, derfor er det satt to ulike begrensinger for hver algoritme på hvilke QR-koder som er med i estimatet:

- Singel BoundingBox, alle QR-koder kap. 3.1.1
- Singel BoundingBox, QR-koder innenfor radius av 500 piksler fra senter
- Singel BoundingBox, de 7 QR-koder nærmest senter
- Snitt av linjer, alle QR-koder kap. 3.1.2
- Snitt av linjer, QR-koder innenfor radius av 500 piksler fra senter
- Snitt av linjer, de 7 QR-koder nærmest senter

Fortest

For å se om det er forskjell på algoritmene under ideelle forhold, er algoritmene først testet på to ideelle bilder. Med tanke på oppløsning og pikselstørrelse, kommer det også frem hvilken presisjon som er relevant å regne videre med. Disse bildene er generert av programvare og har derfor ingen påvirkning fra kamera eller objektiv. De ideelle bildene har lik oppløsning som testbildene som er brukt i hovedtesten.

Resultatet av fortesten er at alle algoritmene estimerer rotasjonen like godt. Rotasjonen ble estimert innenfor et område på ±0,05° for de ulike algoritmene og bildene.

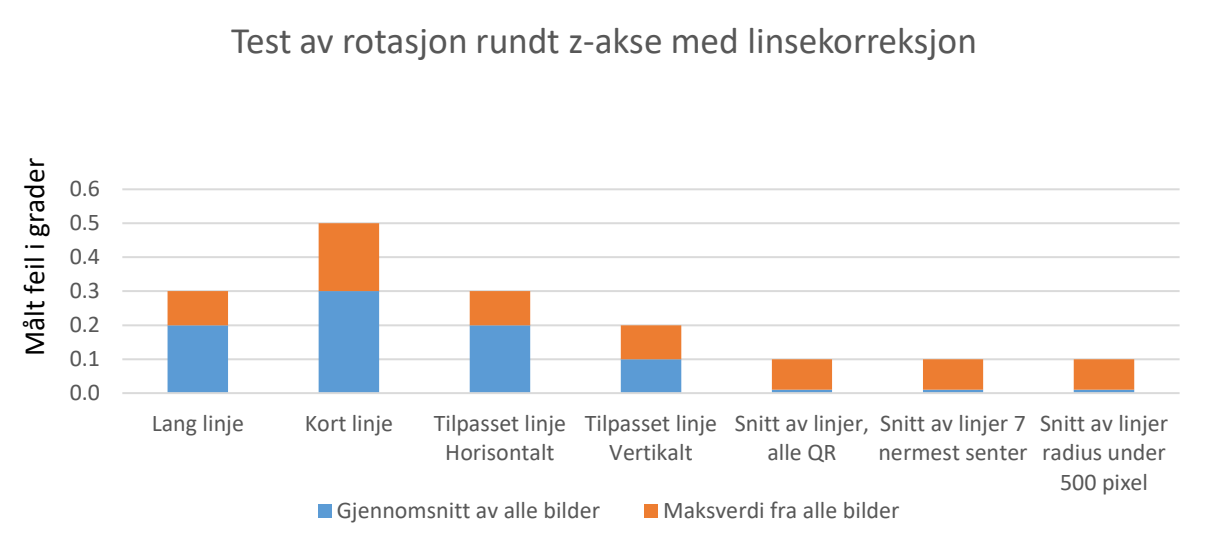
Hovedtest

Basert på resultatet fra fortesten, blir alle algoritmene testet videre og det regnes med en oppnåelig presisjon på ±0,05°. Det regnes derfor med en desimal i resultatene.

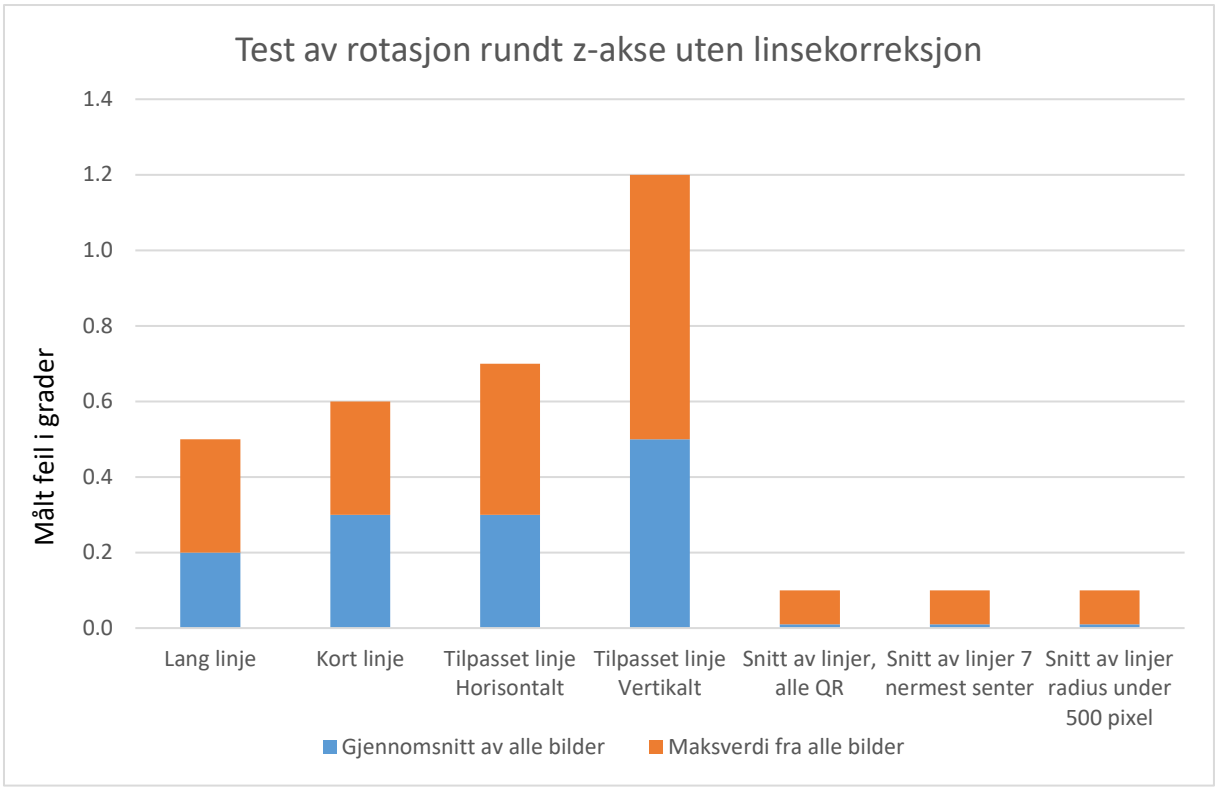
Resultatene i hovedtesten viser at både Singel BoundingBox og Multi BoundingBox ved hyppige tilfeller har store avvik. Disse algoritmene blir heretter sett på som ubrukelige til formålet, og resultatene er derfor tatt ut av diagrammene i dette kapittelet.

Resultatet for bildene med korreksjon for linseforvrengning er presentert i Figur 31. For de som ikke er korrigert for linseforvrengning er resultatene i Figur 32. Resultatene viser at gjennomsnitt av linjene i hver QR-kode gir både laveste maksimal og gjennomsnittlig feil. Med en desimal i resultatet, er det ingen målbar gjennomsnittlig feil. Den maksimale detekterte feilen er 0,1°. Dette gjelder både med og uten korreksjon for linseforvrengning.

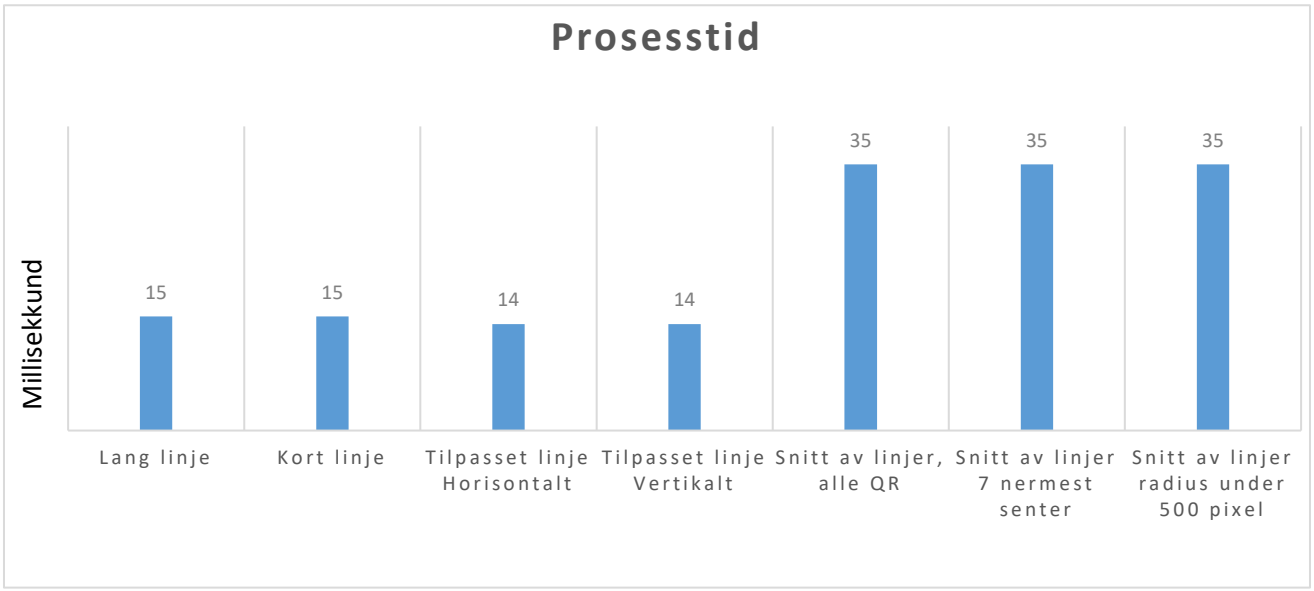
Feilen for snitt av linjer avrundes til null, og det er da ingen forbedring ved å begrense til de QR-kodene med minst avstand fra senter. Prosesstiden til de forskjellige algoritmene viser at Snitt av linjer bruker ca. 2,5 ganger tiden til de andre algoritmene, se Figur 33.



Figur 31. Resultater for hver testet algoritme i de seks bildene hvor det er korrigert for linseforvrengning. Her er maksimalt funnet avvik i alle bildene presentert som oransje søyler. De blå søylene viser gjennomsnittlig avvik i alle bildene. Avviket er oppgitt som absoluttverdi og i enhet grader.



Figur 32. Resultater for hver testet algoritme i de seks bildene hvor det ikke er korrigert for linseforvrengning. Her er maksimalt funnet avvik i alle bildene presentert som oransje søyler. De blå søylene viser gjennomsnittlig avvik i alle bildene. Avviket er oppgitt som absoluttverdi og i enhet grader.



Figur 33. Prosesstid for algoritmene for å finne rotasjon. Tiden er målt i millisekund. Tidene er bare sammenliknbare med hverandre og ikke med andre prosesstider nevnt i denne rapporten.

4.3 Test av algoritmer for å fastsette pikselstørrelsen i romkoordinater

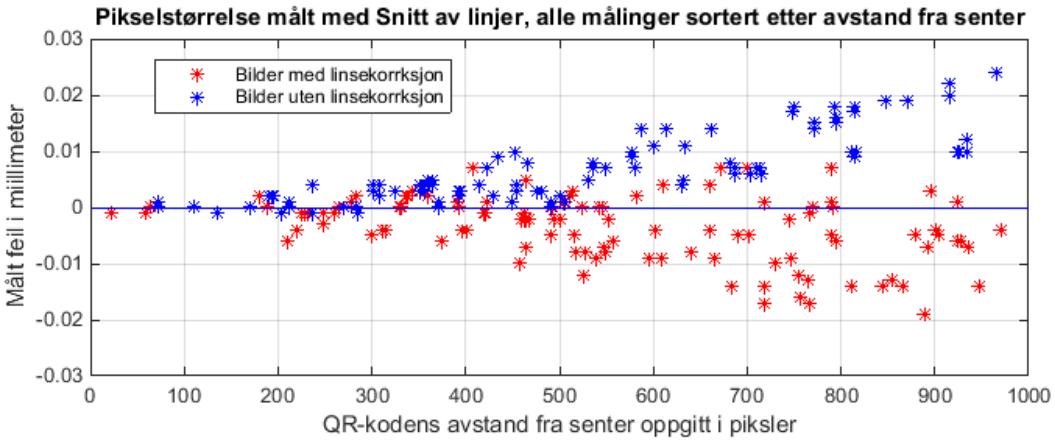
På samme måte som for rotasjonstesten, er også testene av algoritmer for å finne pikselstørrelsen delt opp i to deler. Målet med algoritmene er å finne størrelsen på en piksel i QR-matrisen sitt koordinatsystem. Den første delen ser på hver QR-kode alene. Den andre delen ser på flere QR-koder om gangen. I begge delene er Testoppsett 1 fra kap. 4.1 brukt.

4.3.1 Pikselstørrelse fra single QR-koder

For å se om plasseringen til QR-kodene har sammenheng med målefeil er det kjørt en test av single QR-koder der algoritmen Snitt av linjer fra kap. 3.2.1 er bruk. Den leverer tre resultater som er snitt av alle fire ytterlinjer, snitt av to diagonaler og snitt av alle seks linjer.

Testen viser først og fremst at de tre resultatene gir de samme svarene med differanse under en mikrometer. Resultatene er derfor kun presentert for snitt av alle seks linjer.

Figur 34 viser plott av målefeilene sortert etter QR-kodens avstand til senter. Det røde plottet viser bilder som har blitt korrigert for linseforvrengning. Her er feilen i hovedsak negativ. Det betyr at målt avstand er mindre enn reell. Det blå plottet viser derimot positive feil, som betyr at målingene er større enn reell lengde. Dette er bilder hvor det ikke er korrigert for linseforvrengning. Det ser ut til å være en økning i feil rundt 600 piksler fra senter.



Figur 34. Målefeil fra algoritmen Snitt av linjer, fra hver QR-kode i alle testbilder. Her er målefeilen langs y-aksen, og avstanden til QR-koden fra senter av bildet langs x-aksen.

4.3.2 Pikselstørrelse fra flere QR-koder

For å finne den beste algoritmen for å fastsette pikselstørrelsen i bildet, er flere algoritmer testet mot hverandre.

Følgende algoritmer er testet:

- Lang linje, horisontalt kap. 3.2.2
- Lang linje, vertikalt kap. 3.2.2
- Kort linje, horisontalt kap. 3.2.3
- Kort linje, vertikalt kap. 3.2.3
- Lengst mulig linje kap. 3.2.4

Resultatene fra test av single QR-koder i kap. 4.3.1 viser at for de QR-kodene som har større avstand fra senter enn ca. 600 piksler, er feilen sterkt økende. Derfor er algoritmen Snitt av linjer delt i to tester, en for alle QR-koder og en for de som er nærmere senter enn 600 piksler.

- Snitt av alle linjer, alle QR kap. 3.2.1
- Snitt av alle linjer, Bare QR-koder innenfor radius av 600 piksler fra senter

Fortest

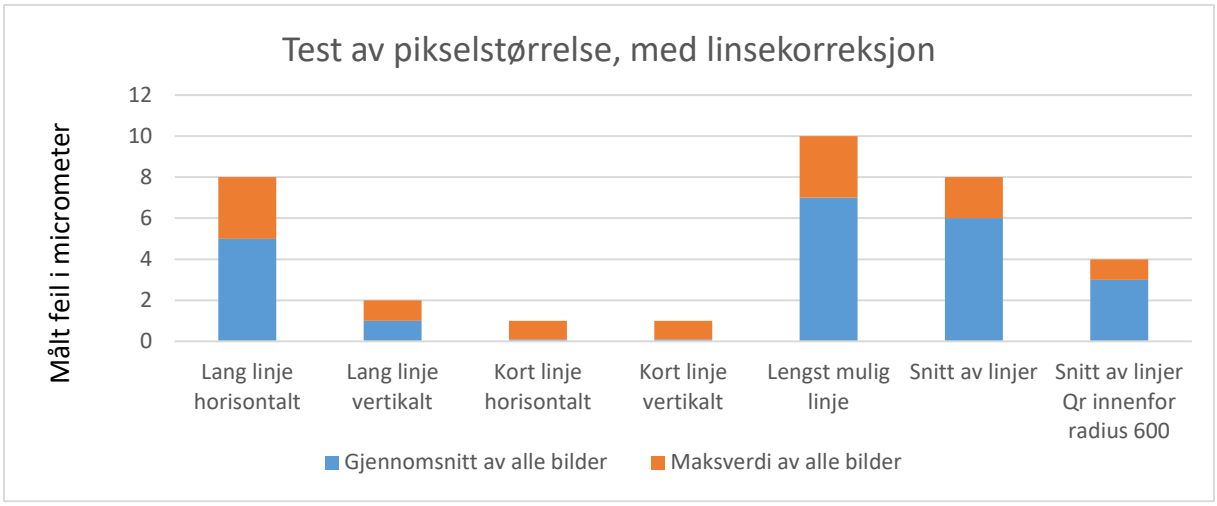
På same måte som i testen for rotasjon, kap.4.2, er også algoritmene her testet på to ideelle bilder. Dette er for å se om algoritmene har et likt utgangspunkt, og for å se hvilken presisjon som er relevant å regne med videre.

Resultatet av fortesten er at alle algoritmene estimerer pikselstørrelsen like godt. Den ble estimert innenfor ±0,6 mikrometer for de ulike bildene og algoritmene.

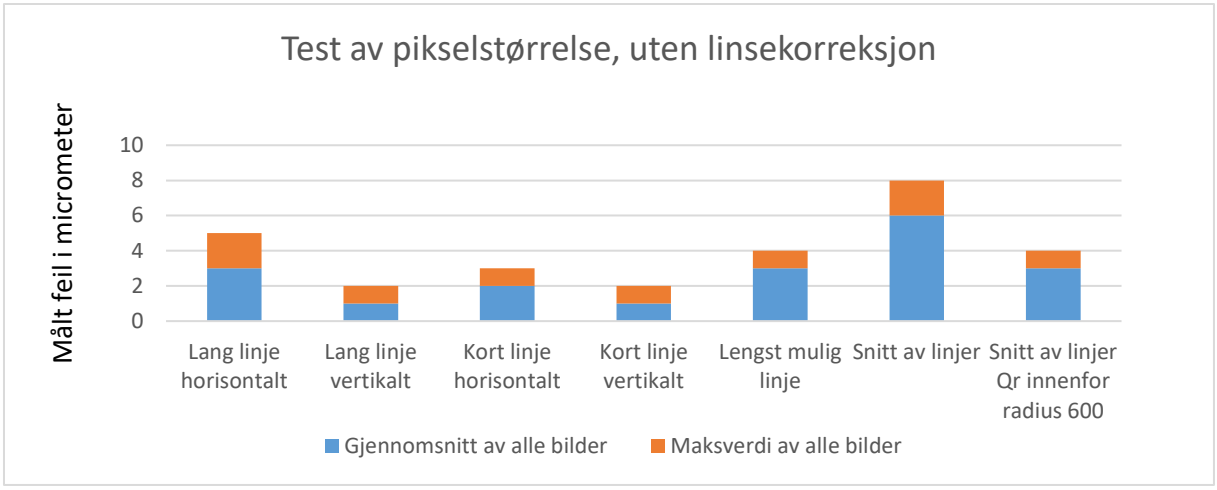
Hovedtest

På bakgrunn av fortesten blir alle algoritmene testet videre i hovedtesten, og det blir sett videre på resultater oppgitt i mikrometer uten desimaler.

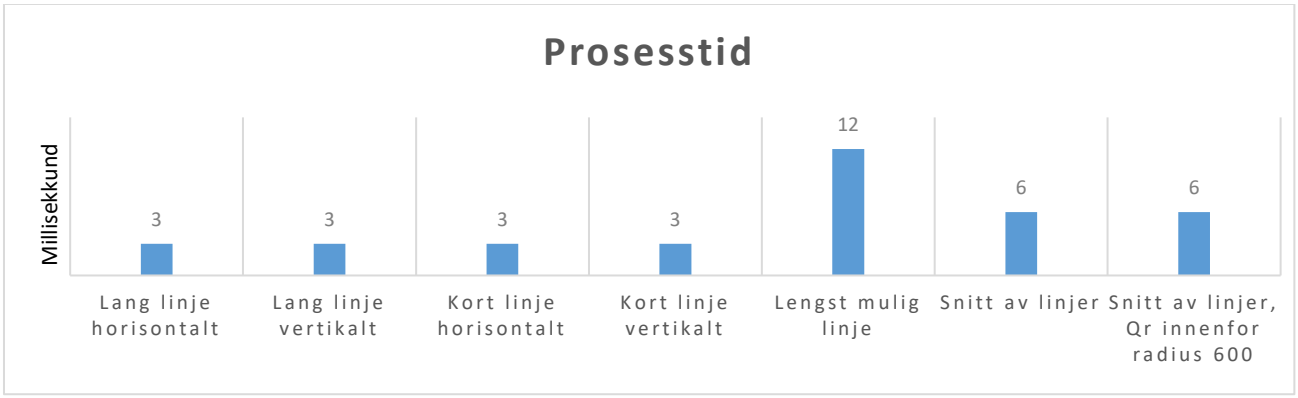
Resultatene for de bildene som er korrigert for linseforvrengning i Figur 35, viser at Kort linje horisontalt og vertikalt gir lavest feil. For bildene som ikke er korrigert for linseforvrengning har Kort linje vertikalt og Lang linje vertikalt lavest feil. Se Figur 36. Prosesstiden for algoritmene er presentert i Figur 37.



Figur 35. Resultater for hver testet algoritme i de seks bildene hvor det ikke er korrigert for linseforvrengning. Her er maksimalt funnet avvik i alle bildene presentert som oransje søyler. De blå søylene viser gjennomsnittlig avvik i alle bildene. Avviket er oppgitt som absolutt verdi og i enhet mikrometer.



Figur 36. Resultater for hver testet algoritme i de seks bildene hvor det er korrigert for linseforvrengning. Her er maksimalt funnet avvik i alle bildene presentert som oransje søyler. De blå søylene viser gjennomsnittlig avvik i alle bildene. Avviket er oppgitt som absolutt verdi og i enhet mikrometer.



Figur 37. Prosesstid for algoritmene for å finne pikselstørrelse. Tiden er målt i millisekund. De er bare sammenliknbare med hverandre, og ikke andre utrekningstider nevnt i denne rapporten

4.4 Test av algoritmer for å estimere senterpunktet til bildet i romkoordinater

Dette kapittelet tar for seg de algoritmene som har som mål å estimere posisjonen til senter av bildet. Resultatene fra rotasjon og pikselstørrelse, i kap 4.2 og 4.3, blir brukt videre her. Dette er parametere som trengs for å kjøre algoritmene i dette kapittelet.

For rotasjon om z-aksen brukes algoritmen Snitt av linjer for alle QR-koder, kap. 3.1.2. For pikselstørrelse brukes algoritmen Kort linje, kap. 3.2.3.

Målet med testene i dette kapittelet er å avgjøre hvilke eller hvilken algoritme som fungerer best til formålet, og om noen av QR-kodene fungerer bedre enn andre til å estimere posisjon. Presisjonen til posisjonsestimatene blir brukt for å sammenligne algoritmene, men den optimale presisjonen blir ikke sett grundig på før i kap. 4.9.

4.4.1 Posisjonsestimater fra single QR-koder

Denne testen er utført for å se på presisjonen av posisjonsestimat fra hver enkelt QR-kode. Testoppsett 1, kap. 4.1 er brukt. Hver av de fire hjørnene til QR-kodene er benyttet slik at posisjonen blir estimert fra fire punkter.

Følgende algoritmer er testet mot hverandre:

- Enkelmetode kap. 3.3.1
- Radiusmetode kap. 3.3.2
- Vinkelmetode kap. 3.3.3
- EPnP kap. 2.7

De fire algoritmene er kjørt hver for seg på hver QR-kode i bildet. Feilen er distansen mellom det estimerte senterpunktet og referansen. Referansen er manuelt målt ut fra hvert bilde med Adobe Photoshop.

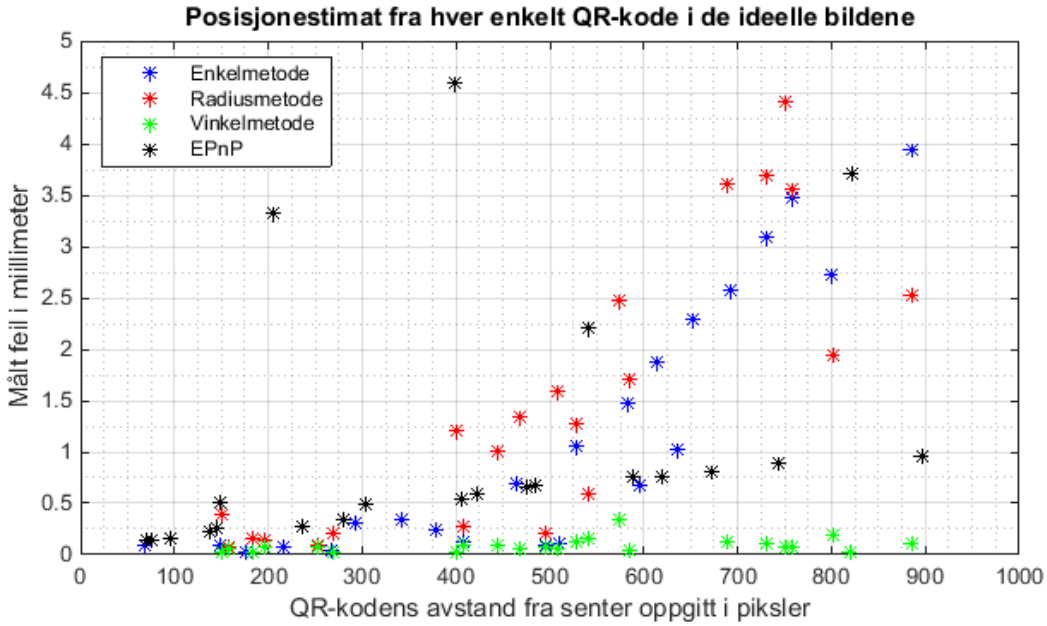
Fortest

På samme måte som i kap. 4.2 og 4.3, er det gjort en fortest på to ideelle bilder for å se hvilke resultater som er oppnåelige.

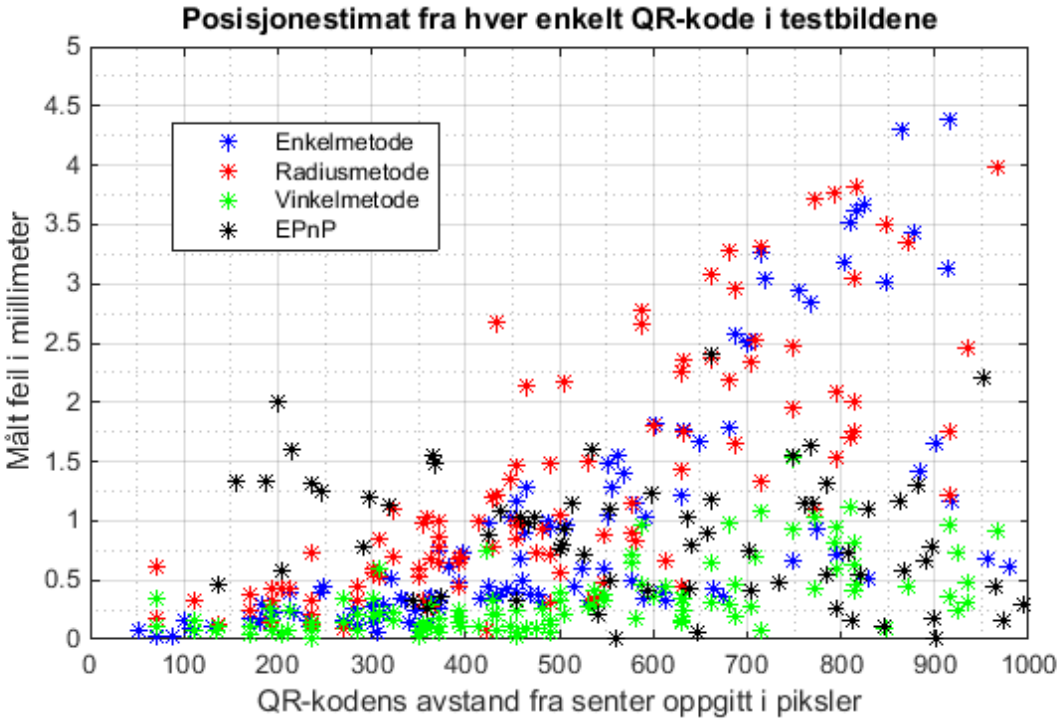
Resultatene for de ideelle i bildene i fortesten er presentert i Figur 38. QR-koder innenfor en avstand på 300 til 400 piksler gir en presisjon på under 0,5 mm og ned mot 0,25 mm. Utover denne avstanden øker feilene mye for algoritmene Enkelmetode og Radiusmetode. Vinkelmetode er mer robust mot store avstander fra senter. EPnP ser ut til å ha mer tilfeldig fordelte feil enn de andre algoritmene.

Hovedtest

Resultatene fra testbildene i hovedtesten er presentert i Figur 39. De viser de samme tendensene som de to ideelle bildene. Med avstand over 400 piksler, øker feilen betraktelig for Enkelmetode og Radiusmetode. For EPnP var det i tillegg en del målinger som overgikk y-aksen på Figur 39. Disse feilene var på mellom 50 og 1000 millimeter.



Figur 38. Posisjonsestimatet for hver enkelt QR-kode i fortesten. Resultatene viser at Radiusmetode og Enkelmetode gir store feil for QR-koder utover 300-400 pikslers avstand fra senter. Vinkelmetode derimot tåler store avstander bedre. EPnP har mer tilfeldige feil. Det ser ut til at feil under 0,25 mm kan være oppnåelig.



Figur 39. Posisjonsestimatet for hver enkelt QR-kode i hovedtesten viser i likhet med de ideelle bildene at for QR-koder med avstand utover 300 til 400 piksler fra senter, blir feilen betraktelig høyere. Spesielt for Enkelmetode og Radiusmetode er dette tilfelle. Også i hovedtesten utmerker Vinkelmetode seg som bedre enn de andre algoritmene. Algoritmen EPnP har noen outliers som overgår y-aksen på figuren. Feilene fra EPnP er mer tilfeldig fordelt enn de andre.

4.4.2 Posisjonsestimat fra flere QR-koder

Denne testen bruker testoppsett 2 fra kap. 4.1. Det blir tatt 3000 bilder av en QR-matrise hvor senterpunktet er kjent og låst fast gjennom hele testen. Hvert av bildene som blir tatt og estimert posisjon fra kalles en måling. Gjennomsnittet og standardavviket av målingene blir brukt for å sammenligne algoritmene. Som forklart i innledningen til kap. 4 blir det lagt størst vekt på et lavt standardavvik.

På bakgrunn av resultatene fra kap. 4.4.1 som viser at QR-koder med avstand under 400 piksler fra senter av bildet har bedre presisjonen de som er lenger fra senter, er kapittelet delt opp i to deler. En del hvor hele bildet er brukt, og en hvor bare QR-koder nærmere enn 400 piksler fra senter er tatt med i beregningen.

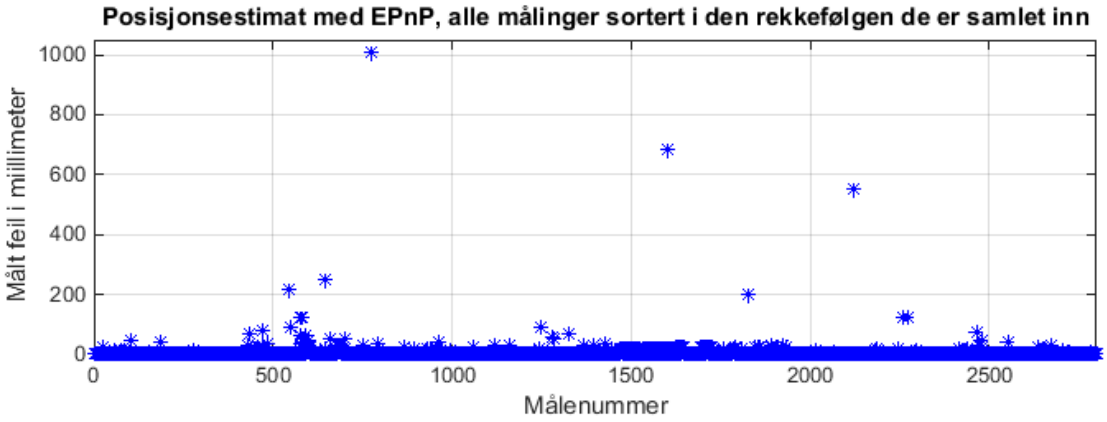
Underveis i testen er et a4-ark lagt over deler av QR-matrisen slik at alt fra 1 til 16 QR-koder har vært synlige på bildet.

Følgende algoritmer er testet mot hverandre:

- Enkelmetode kap. 3.3.1
- Radiusmetode kap. 3.3.2
- Vinkelmetode kap. 3.3.3
- EPnP kap. 2.7

EPnP er en algoritme som kjøres direkte fra OpenCV. Den estimerer kameraet sin posisjon, og ikke senter av bildet slik som de andre algoritmene. Så lenge kameraet står normalt på QR-matrisen, vil de todimensjonale x- og y-koordinatene til kameraet stemme med senter av bildet. Denne algoritmen er tatt med for å sammenligne det som er konstruert i prosjektet med en algoritme som er uavhengig.

Resultatene fra algoritmen EPnP viser at estimatene har mange outliers, eller store avvik opp mot 1000 millimeter, dette er vist i Figur 40. Mellom disse store avvikene er estimatene på et normalt nivå sammenlignet med de andre algoritmene. Det ser ikke ut til å være noe system i når disse feilene oppstår. Dette gjør algoritmen ubrukelig til formålet, men den er likevel tatt med som et sammenligningsgrunnlag i dette kapittelet. For å få et sammenligningsgrunnlag, er alle estimatene fra EPnP med feil over 10 millimeter tatt ut av resultatene.



Figur 40. Estimatene fra algoritmen EPnP har hyppige feil opp mot 1000 millimeter. Ved å luke ut de feil som overstiger 10 millimeter, kan estimatene likevel brukes som et sammenligningsgrunnlag.

Posisjonsestimat fra hele bildet

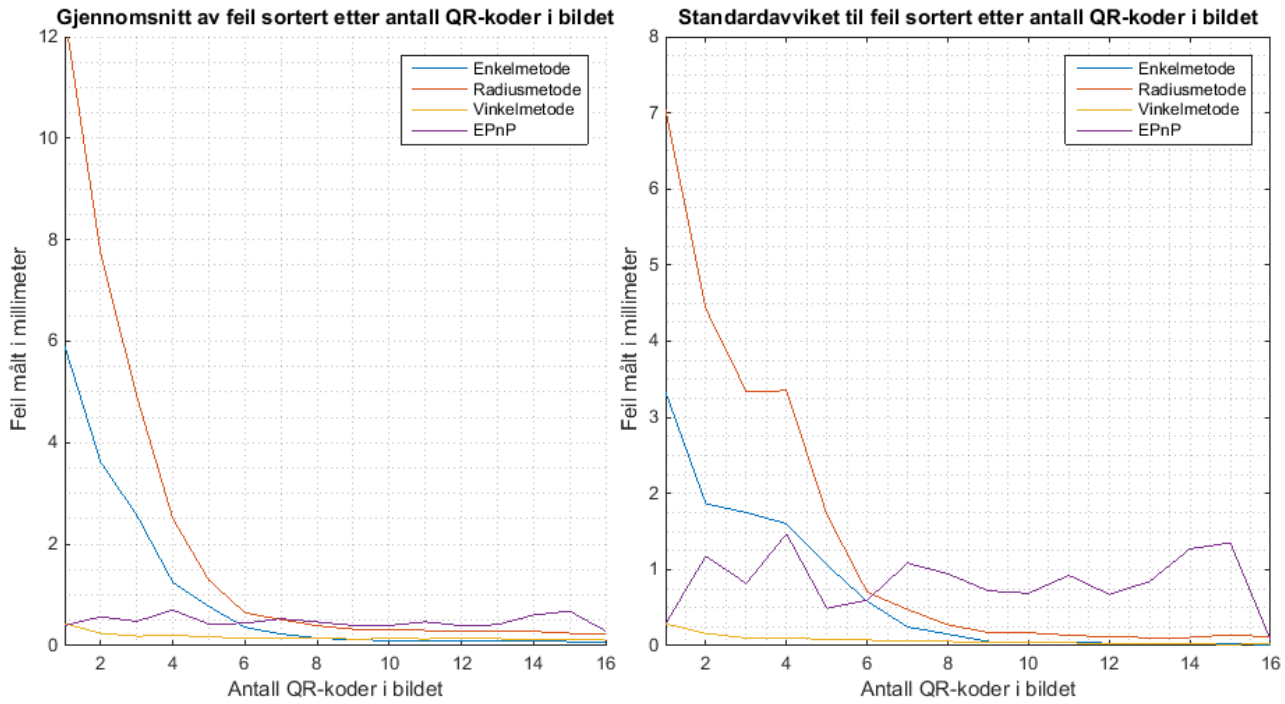
I denne delen av testen er posisjonen estimert med alle detekterte QR-koder som inngangsparameter. Resultatene er presentert som gjennomsnitt og standardavvik av 3000 bilder, eller målinger. Det er detektert mellom 1 og 16 QR-koder i hver måling.

Figur 41 viser resultatene sortert etter antall QR-koder detektert i bildet. For å se detaljene bedre, viser Figur 42 de samme resultatene, men her er det zoomet inn på området med mer enn 6 QR-koder detektert.

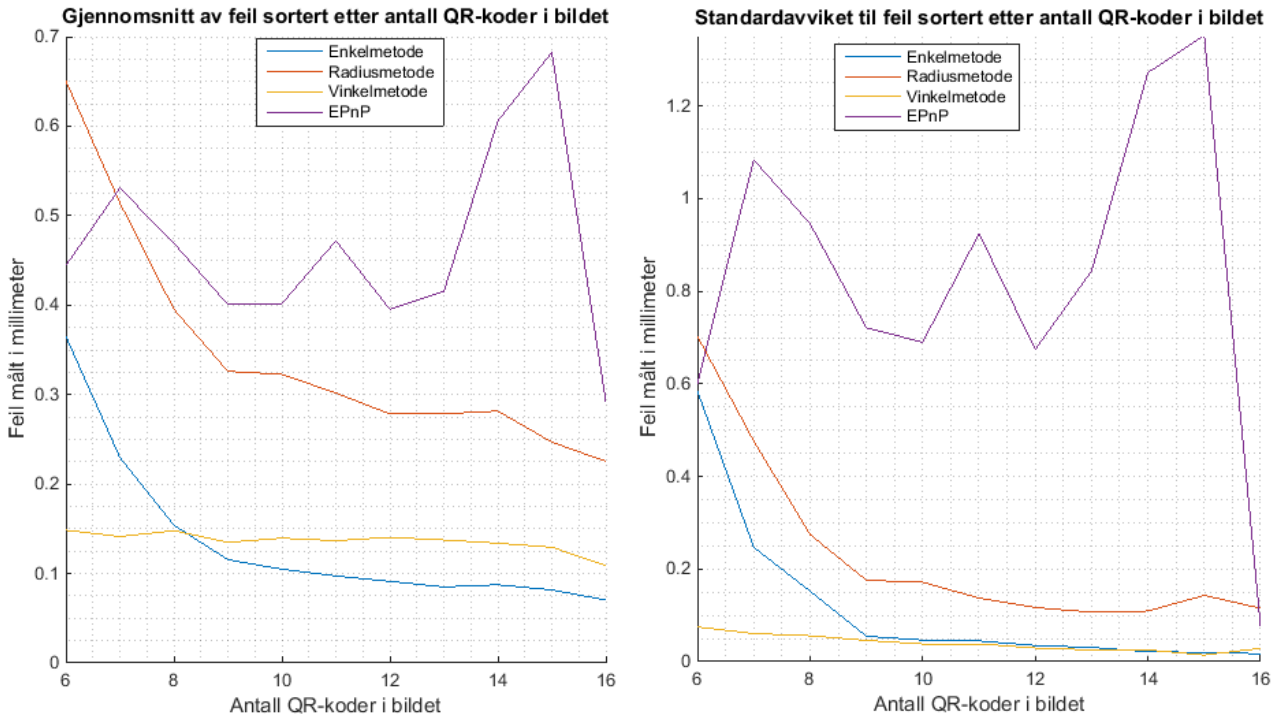
Vinkelmetode og EPnP er de som er minst avhengige av antallet QR-koder. Radiusmetode estimerer dårligere enn Vinkelmetode og Enkelmetode, men den estimerer bedre enn EPnP ved stort antall QR-koder. Vinkelmetode fungerer bedre enn kontrollmetoden EPnP uansett antall QR-koder. Ved stort antall QR-koder, funger alle algoritmene bedre enn EPnP. Prosesstiden til de fire algoritmene er presentert i Tabell 3.

Tabell 3. Prosesstid til algoritmer for å estimere senterpunkt. Tidene er relative til hverandre, men kan ikke sammenlignes med andre tider i rapporten.

Algoritme	**Prosesstid**
Vinkelmetode	10 ms
Radiusmetode	6 ms
Enkelmetode	10 ms
EPnP	3 ms



Figur 41. Resultatene viser at Vinkelmetode og EPnP er minst avhengig av antall QR-koder. Vinkelmetode fungerer bra selv med få QR-koder detektert. Standardavvik og gjennomsnitt er estimert for hvert heltall på x-aksen, interpoleringen mellom tallene er bare for å se sammenhengen i de ulike algoritmene.



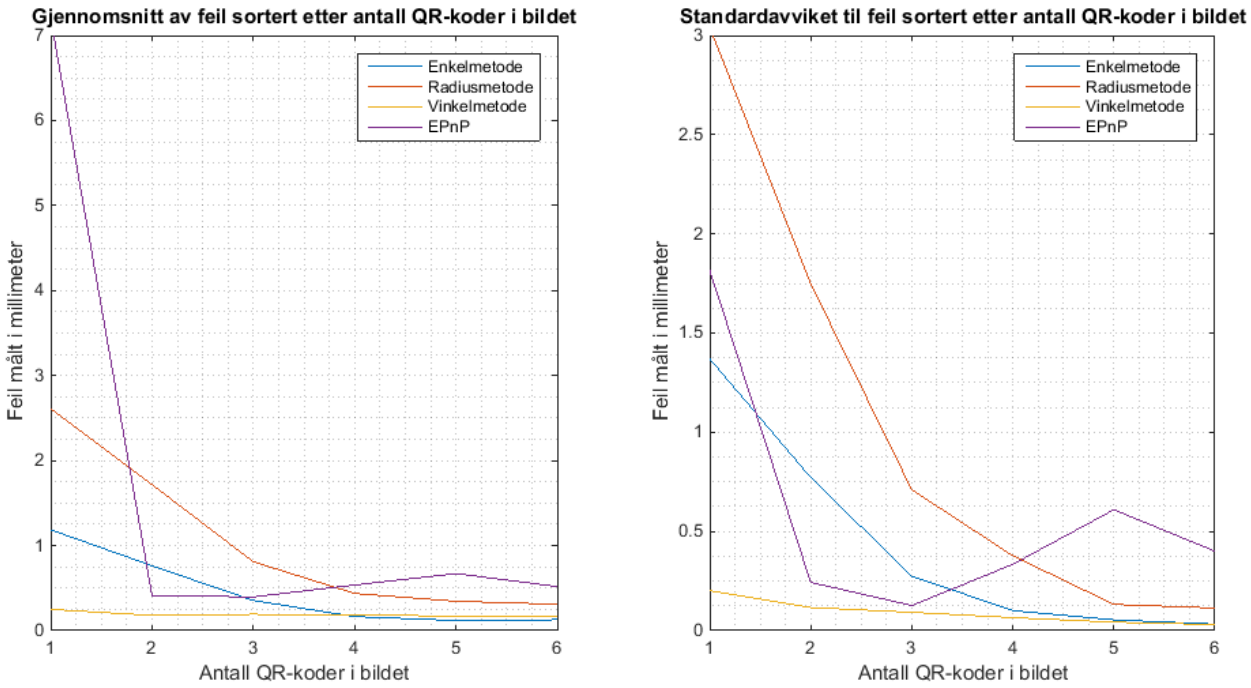
Figur 42. Ved å zoome inn på grafen, er det tydelig at Vinkelmetode fungerer best når det er få antall QR-koder, men med flere QR-koder får Enkelmetode like god, eller bedre presisjon.

Posisjonsestimat fra QR-koder innenfor 400 piksler

Denne delen av forsøket fungerer på samme måte som posisjonsestimatene fra hele bildet. Forskjellen er at det kun er de QR-kodene som er innenfor en radius av 400 piksler fra senter av bildet som er tatt med i beregningen. Her er mellom en og seks QR-koder detektert.

Figur 43 viser resultatene. Testen viser svært like resultater som for testen av hele bildet. Det er Enkelmetode og Vinkelmetode som utmerker seg best. Den oppnåelige presisjonen er ikke vesentlig bedre enn for hele bildet. Resultatene viser ingen videre grunn til å begrense seg til QR-koder med avstand under 400 piksler fra senter.

Videre i kapittelet blir Enkelmetode og Vinkelmetode brukt uten begrensing på hvilke QR-koder som skal være med i estimatet.

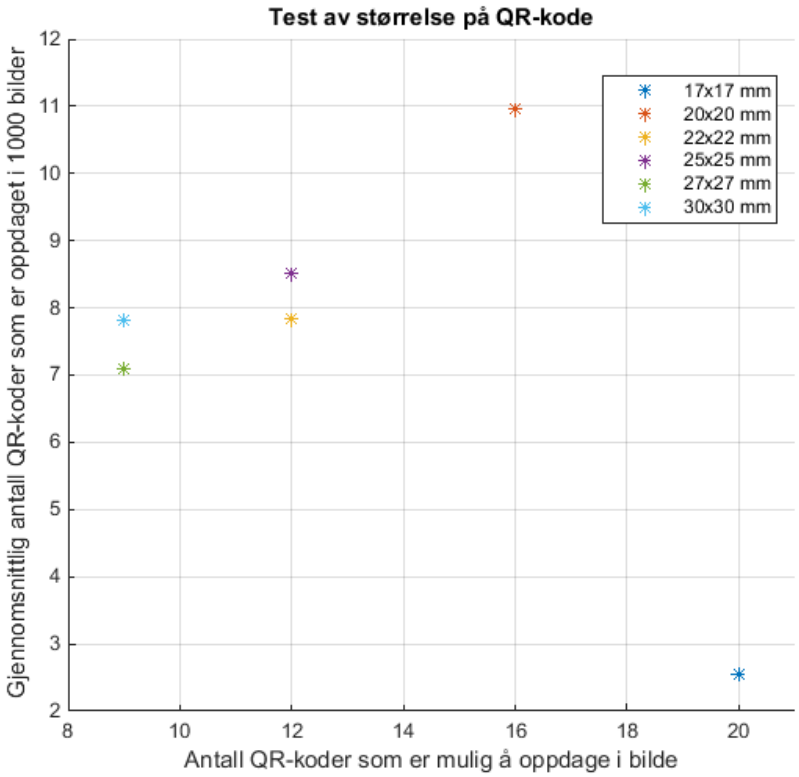


Figur 43. Resultatene fra test med bare QR-koder nærmere enn 400 piksler fra senter viser ingen vesentlig forskjell fra resultatene for hele bildet i Figur 41 og Figur 42. Standardavvik og gjennomsnitt er estimert for hvert heltall på x-aksen, interpoleringen mellom tallene er bare for å se sammenhengen i de ulike algoritmene.

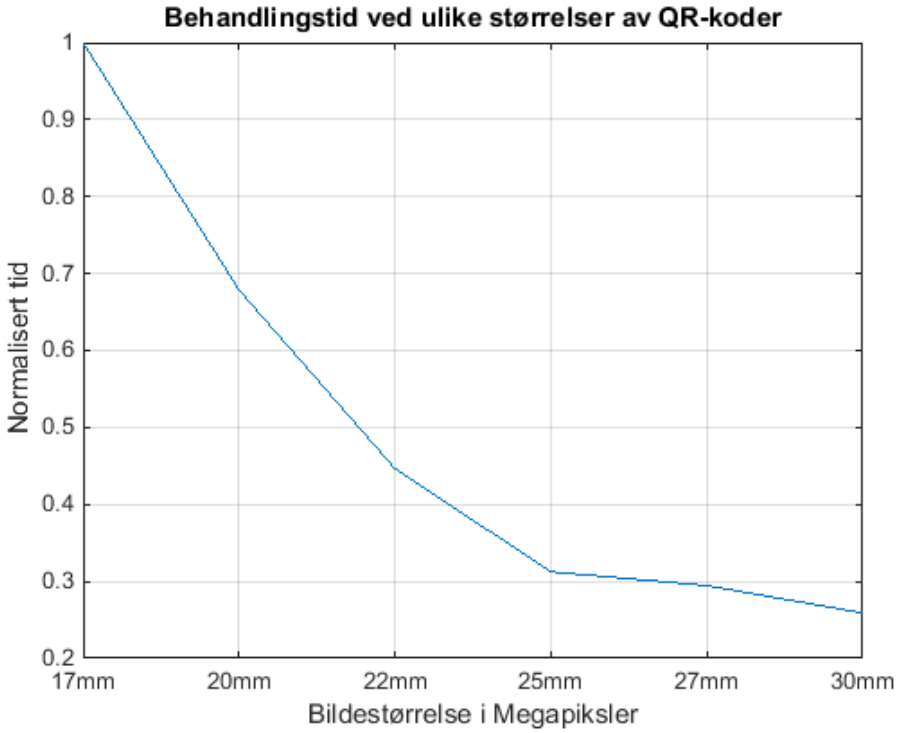
4.5 Test av ulike størrelser og antall QR-koder

Desto flere QR-koder som er detektert, desto flere punkter er det å estimere senterpunktet til bildet fra. Denne testen tar for seg et utvalg størrelser av QR-koder med sider fra 17 mm til 30 mm. Testen er satt opp med Testoppsett 2 fra kap. 4.1. Det er tatt 1000 bilder for hver størrelse, og alle er tatt i en avstand på 200 mm fra QR-matrisen. For de største QR-kodene er det mulig å detektere ni QR-koder i bildet. For de minste er det mulig å detektere 20. Ønsket resultat er flest mulig oppdagede QR-koder.

Figur 44 viser resultatene fra testen. Her er hver av størrelsene plottet med en egen farge. Y-aksen viser gjennomsnittlig hvor mange QR-koder som er detektert. X-aksen viser hvor mange som er mulig å detektere. For størrelse 20 mm er flest QR-koder detektert. Dess større QR-kodene er, dess større andel av de som er mulig å oppdage blir oppdaget. Når QR-kodene blir så små som 17 mm, er det svært få som blir detektert. I Figur 45 er behandlingstiden presentert. Større QR-koder skannes raskere enn de som er mindre.



Figur 44. Y-aksen viser gjennomsnittlig antall detekterte QR-koder for hver størrelse. X-aksen viser antall QR-koder som er mulig å detektere. Ved bruk av større QR-koder, blir en større andel detektert. Med mindre QR-koder er det mulig å detektere flere. 20 millimeter gir størst antall detekterte. Ved 17 millimeter er svært få detektert.



Figur 45. Behandlingstiden for å skanne bildet er avhengig av størrelsen på QR-kodene. Tidene er normaliserte slik at den største er 1.

4.6 Test av linsekorrigering og ulike objektiv på kameraet

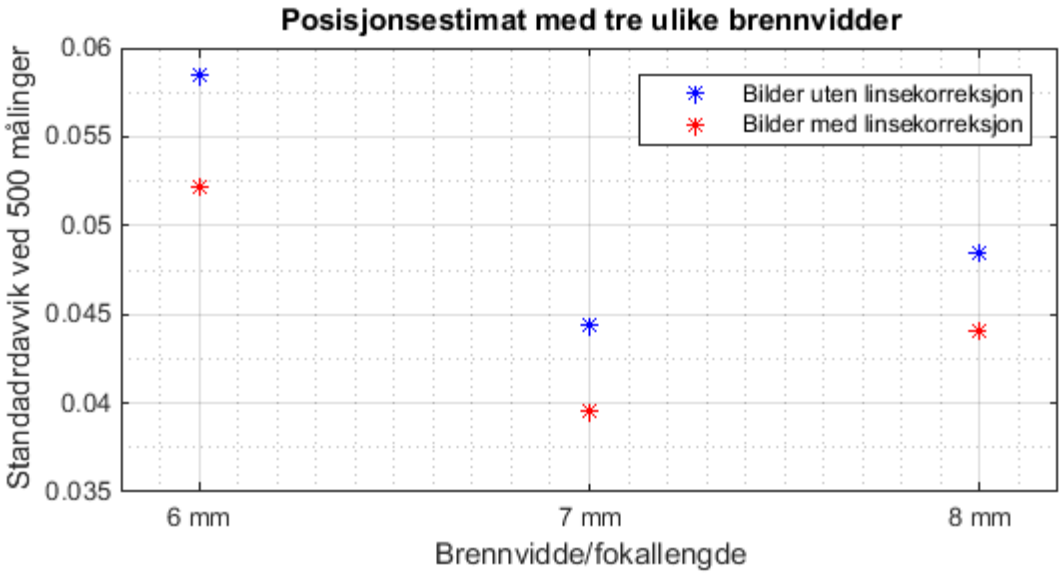
Dette forsøket er delt opp i to tester for å se på to faktorer rundt valg av objektiv til kameraet. Testoppsett 2 fra kap. 4.1 er brukt, og som tidligere blir standardavviket til flere målinger brukt som sammenligningsgrunnlag.

Normalobjektivet til kameraet har brennvidde på 9 mm. Kameraet er utstyrt med et zoomobjektiv hvor brennvidden kan justeres fra 1,2 mm til 8 mm. Dette gir et arbeidsområde med bare vidvinkelperspektiv. Det som er ønskelig å finne ut i den første testen, er om presisjonen til posisjonsestimatet lar seg påvirke av avvik fra normalobjektivet til kameraet. I neste test er det ønskelig å se på hvilken effekt linsekorrigering kan ha på estimatene, se kap. 2.2 om linseforvrengning og kamerakalibrering.

Test av ulike brennvidder

Det er tatt bilder med 6,7 og 8 mm brennvidde på objektivet. Fra denne testen er det ønskelig å se påvirkningen fra linseforvrengingen, og fra minst mulig andre parametere. Derfor er kameraet plassert 200 mm fra QR-matrisen når 8 mm brennvidde er brukt, og når de mindre brennviddene er brukt, er kameraet flyttet nærmere matrisen slik at bildene viser de samme motivet. Dette gjør at det er like mange QR-koder med lik størrelse i alle bildene. Da vil ikke antall QR-koder, og plasseringen de har i bildet ha påvirkning på resultatene.

Figur 46 viser standardavviket fra estimatene av senterposisjonen med Vinkelmetode. Både med og uten linsekorreksjon er resultatene svært like for de tre brennviddene.



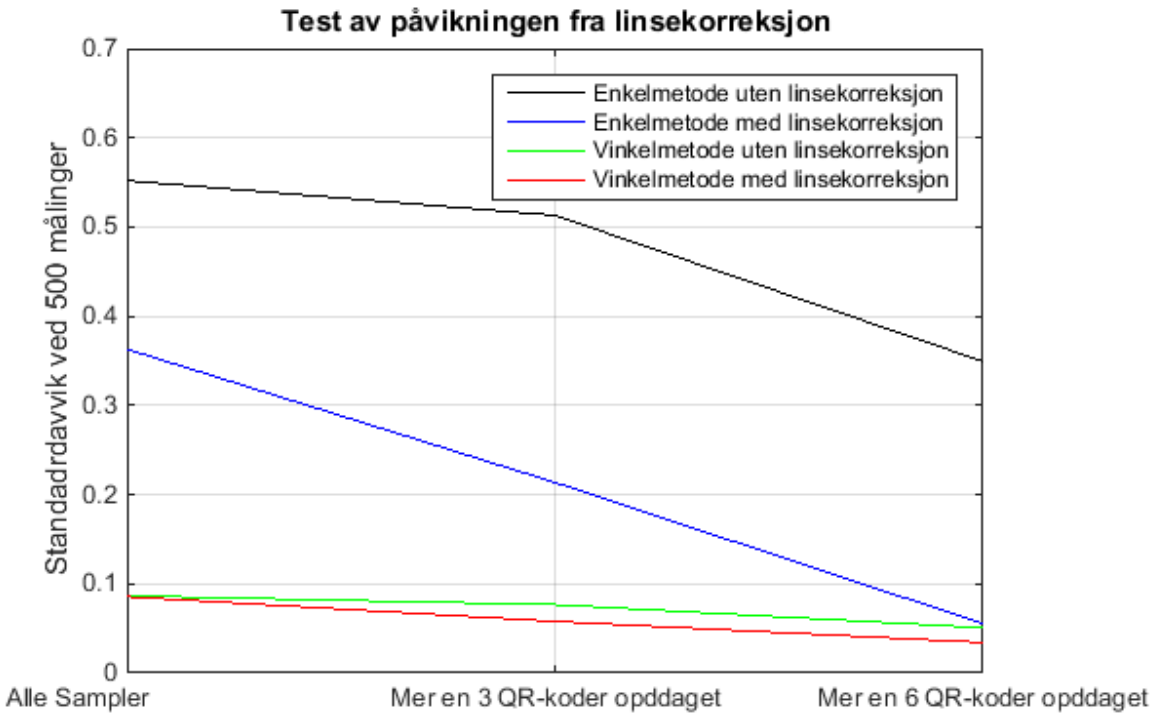
Figur 46. Standardavviket til estimater av posisjon med tre forskjellige brennvidder på objektivet, både med og uten linsekorreksjon. Resultatene er svært like hverandre, med en forskjell på 0,02 mm.

Effekt på estimater av posisjon ved bruk av linsekorreksjon

Det er utført en test av de to algoritmene, Enkelmetode og Vinkelmetode, med 8 mm brennvidde på objektivet. Her er posisjonsestimatene sortert etter hvor mange QR-koder som er detektert. Det er en kategori for alle målinger. En kategori for hvor det er detektert mer enn tre QR-koder, og en hvor det er detektert mer enn seks QR-koder. Posisjonen er estimert både med og uten linsekorreksjon. Dette er for å se hvilken påvirkning linsekorreksjonen har på de to algoritmene, og om den eventuelle påvirkningen har sammenheng med antall QR-koder estimatet er gjort fra.

Figur 47 viser resultatene. Enkelmetode har mye større påvirkning fra linsekorreksjonen enn hva Vinkelmetode har. Vinkelmetode fungerer svært godt både med og uten linsekorreksjon. Felles for begge algoritmene er at desto større presisjonen blir, desto mer blir også resultatet forbedret med linsekorreksjon. Forbedringen for Vinkelmetode er på 1% ved de fleste tilfeller og opp mot 30% når presisjonen øker. For Enkelmetode er forbedringen på 30% ved de fleste tilfeller og opp mot 85% ved høy presisjon.

I tillegg til denne testen har resultatene i testene fra kap. 4.4, 4.7, 4.8 og 4.9 blitt lagret både med og uten linsekorreksjon. Plottene i de respektive kapitlene viser ikke dette, men tendensene er like som det Figur 47 viser. Ved bruk at Enkelmetode er det en del å hente på linsekorreksjon, men for Vinkelmetode er forbedringen minimal.



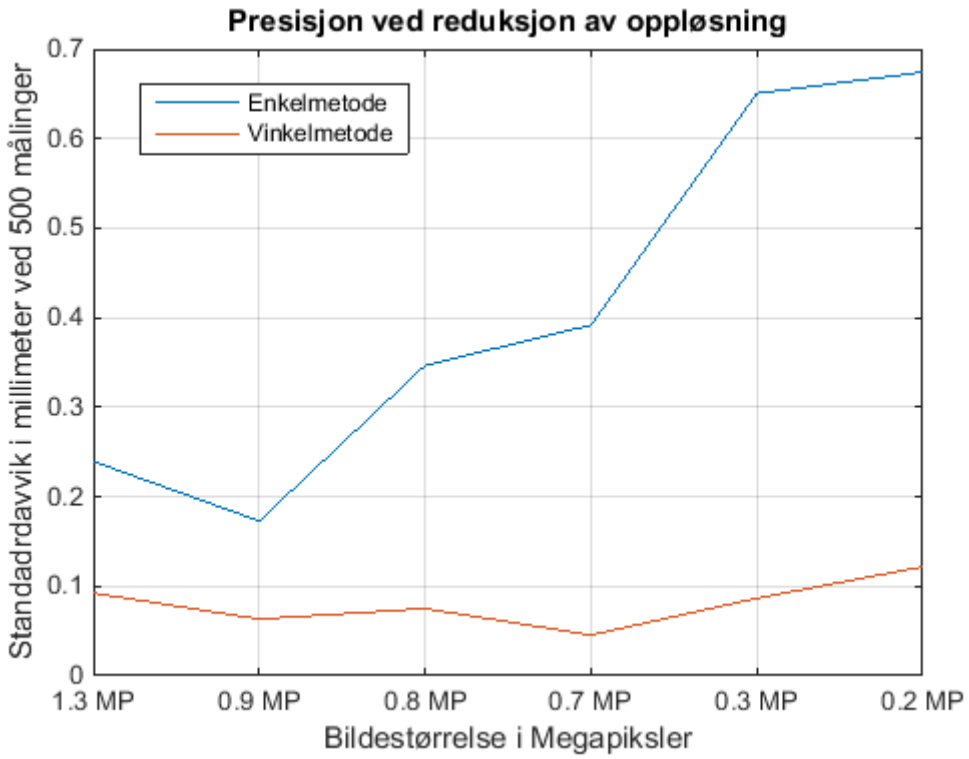
Figur 47. Standardavvik fra posisjonsestimater med Enkelmetode og Vinkelmetode. Begge med og uten linsekorreksjon. Det er gjort tre estimater med betingelse på hvor mange QR-koder som minst må være detektert i bildet. Enkelmetode har mest å hente på linsekorreksjon. Forbedringen med linsekorreksjon blir større jo mer presist estimatet er. Det er interpolert mellom de tre kategoriene for bedre å se økningen i forbedring ettersom presisjonen øker.

4.7 Test av påvirkning på posisjonsestimat ved reduksjon av oppløsning

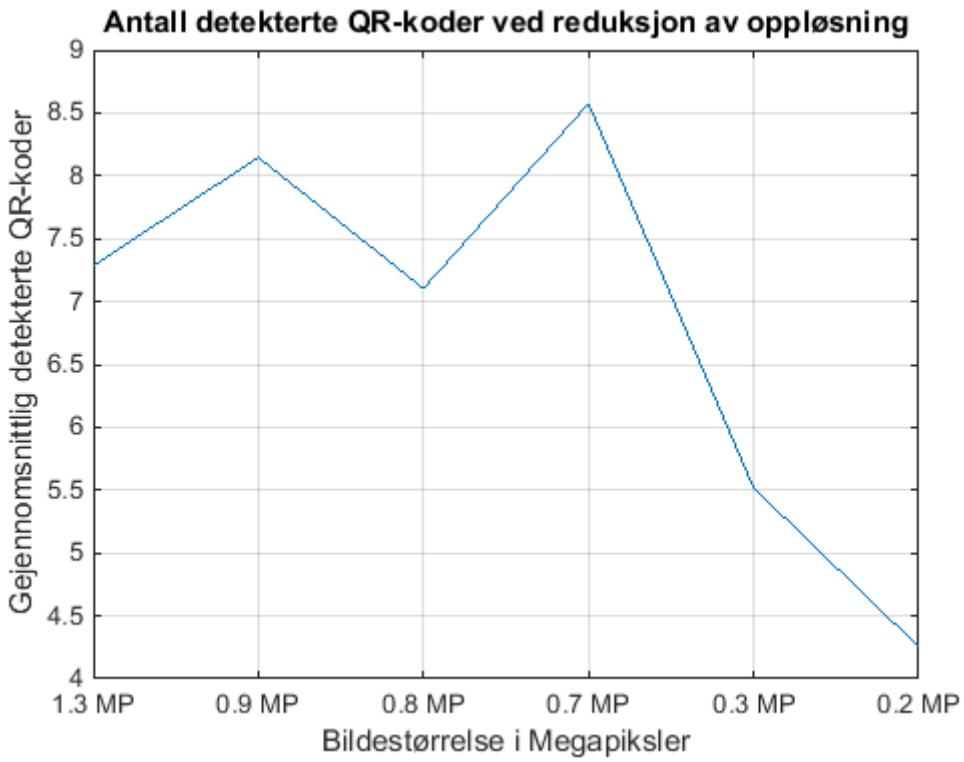
Denne testen er gjort for å se på hvilken påvirkning det har å redusere oppløsningen til kameraet. Dette både i forhold til presisjon i estimatet av senterposisjon, og i forhold til hvor mange QR-koder som blir detektert når bildet blir skannet. Det blir også sett på om behandlingstiden har sammenheng med oppløsningen. Testen er gjort med Testoppsett 2 fra kap. 4.1. Posisjonen er estimert, og antallet QR-kodene er lagret for bilder med oppløsning fra 0,2 til 1,3 megapiksler(MP) der 1,3 MP er maksimal oppløsning. Posisjonene er estimert med algoritmene Enkelmetode og Vinkelmetode.

Resultatene for posisjonsestimatene er presentert i Figur 48. Enkelmetode fungerer best ved 0,9 MP, og den fungerer dårligere ettersom oppløsningen går ned. Vinkelmetode har bedre estimater med lavere oppløsning. Her er det beste estimatene på 0,7 MP. Dette kan ha sammenheng med resultatene i Figur 49. som viser antall detekterte QR-koder. Grafen viser at ved å gå ned på oppløsningen, øker antall detekterte QR-koder. Med oppløsning under 0,7 MP, synker antallet igjen.

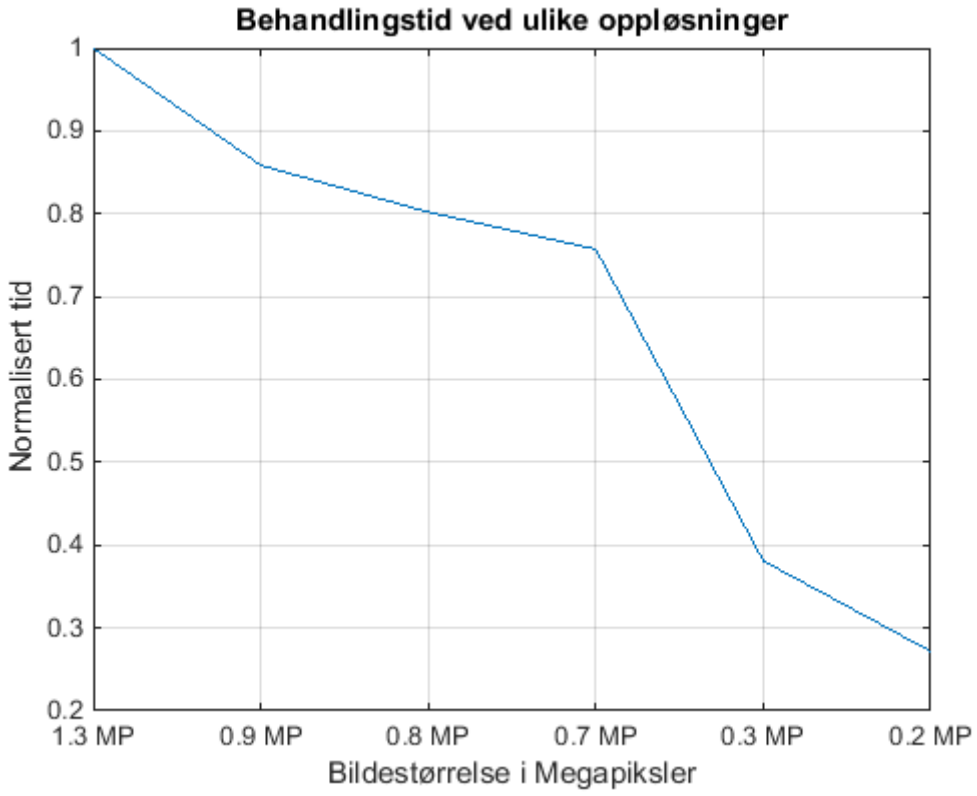
Figur 50 viser at det er tidsbesparende å redusere oppløsningen før bildet skannes av ZBar.



Figur 48. Standardavvik fra posisjonsestimater med reduksjon i oppløsning. Enkelmetode får stor påvirkning fra å redusere oppløsningen. Vinkelmetode er robust mot reduksjonen og kan til og med fungere bedre. Det er sannsynlig at forbedringen har med antall QR-koder som er detektert å gjøre, se Figur 49. Standardavviket er estimert for hver oppløsning på x-aksen. Interpoleringen mellom oppløsningene er bare for å se sammenhengen i de ulike algoritmene.



Figur 49. Antall detekterte QR-koder ved reduksjon av oppløsningen. ZBar detekterer flere QR-koder når oppløsningen blir redusert til 0,7 megapiksler. Ved større reduksjon minker deteksjonsraten igjen. Gjennomsnittet er estimert for hver oppløsning på x-aksen. Interpoleringen mellom oppløsningene er bare for å se sammenhengen.

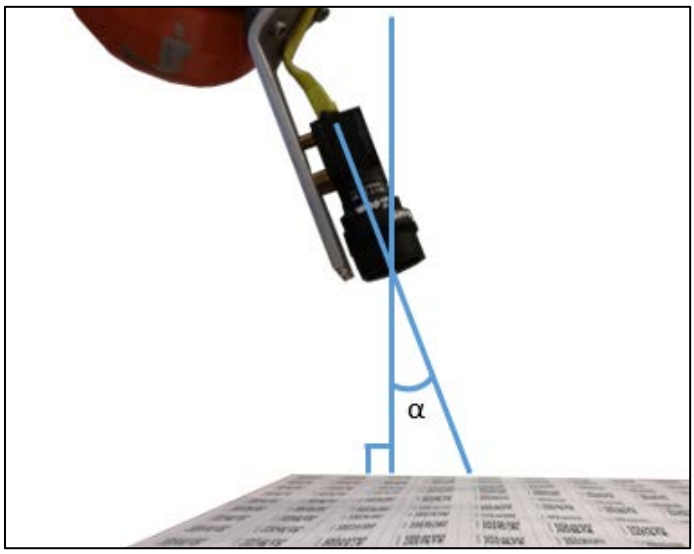


Figur 50. Tiden ZBar bruker på å skanne bildet er avhengig av oppløsningen. Tiden er normalisert slik at 1 er høyeste verdi.

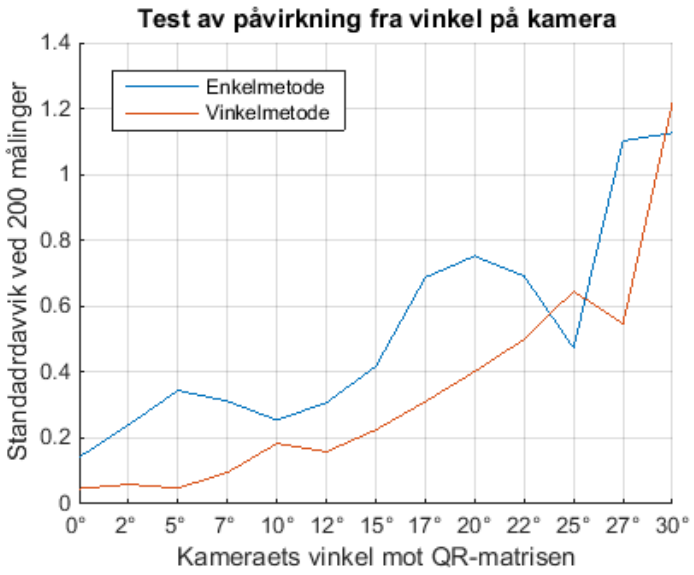
4.8 Test av påvirkning fra vinkel på kameraet

Prosjektet har fra starten av hatt en betingelse, og den er at kameraet skal stå normalt på QR-matrisen. Dette kapittelet tar for seg en test hvor målet er å utfordre denne betingelsen. Testoppsett 2 fra kap. 4.1 er brukt. Vinkelen α i Figur 51 er 0° når kameraet står normalt på matrisen. Denne vinkelen økes i steg på to og tre grader opp til 30°. Standardavviket til posisjonen blir estimert for hvert steg.

Algoritmene Enkelmetode og Vinkelmetode er brukt til å estimere posisjonen. Resultatene i Figur 52 viser at Vinkelmetode fungerer best over det meste av spekteret. Når vinkelen α økes fra 0 til 5°, er estimatet uendret for denne algoritmen, men for Enkelmetode dobler feilen seg. Ved α større enn 12° til 15°, øker feilen mye for begge algoritmene.



Figur 51. Vinkelen α er begrenset til å være 0° tidligere i prosjektet. I dette kapitlet utfordreres denne betingelsen ved å øke α fra 0° til 30°.



Figur 52. Resultatene fra posisjonsestimat med vinkel på kameraet viser presise estimater for Vinkelmetode når vinkelen er under 5°. Utover 12-15°, øker feilen drastisk for begge algoritmene. Over 20° spiller fokusplanet en rolle og gjør at presisjonen går opp og ned etter hvor mange QR-koder som blir tydelige i bildet. Standardavviket er estimert for hver av de oppgitte vinklene på x-aksen, interpoleringen mellom de er bare for å se sammenhengen i de ulike algoritmene.

4.9 Test av presisjonen til systemet

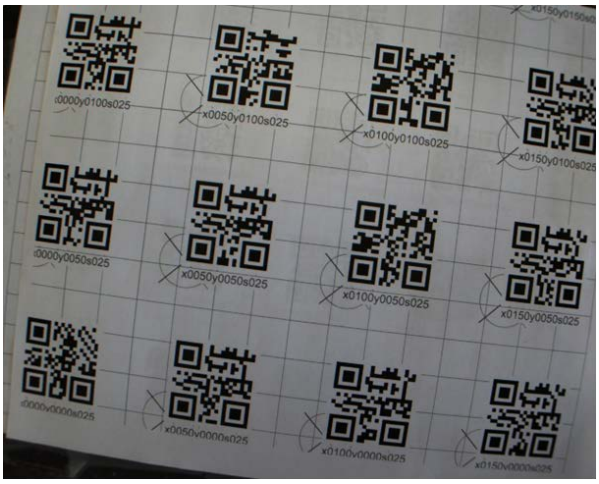
I de tidligere testene i rapporten har presisjonen gitt sammenligningsgrunnlag for den aktuelle testen. Målet med testen i dette kapittelet er å fastsette hva som er oppnåelig presisjon for systemet. Testen er gjort med testoppsett 2 fra kap. 4.1. Algoritmene Enkelmetode og Vinkelmetode er testet både med 1,3 og 0,7 megapiksler(MP) oppløsning på bildet. Resultatene fra de tidligere testene i kap. 4 er brukt for å oppnå den best mulige presisjonen.

Posisjonen til senter av bildet er estimert fra 500 bilder, og standardavviket brukes som parameter. Ved å ta utgangspunkt i normalfordelingen, er presisjonen ±3σ, der σ er standardavviket. Presisjonen ved 3σ vil dekke 99,7% av alle posisjonsestimatene.

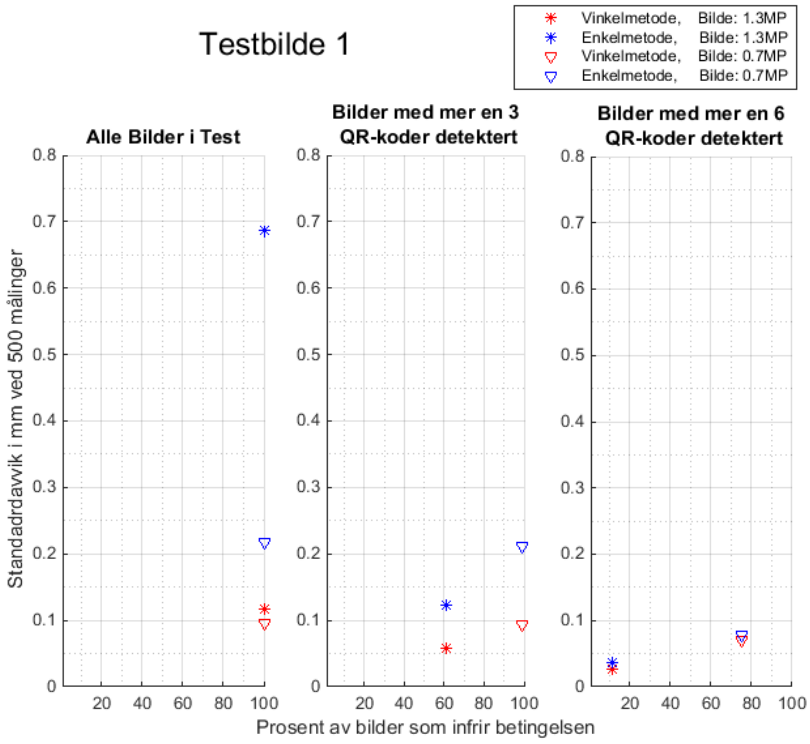
Testen er utført tre ganger med ulik plassering over QR-matrisen, se testbildene i Figur 53, Figur 56 og Figur 57. Posisjonen er estimert tre ganger med ulike betingelser for hvert bilde. Det første posisjonsestimatet er funnet basert på alle målinger, og det neste estimatet på de målinger hvor det er detektert mer enn tre QR-koder. Det siste estimatet er funnet på bakgrunn av de målinger hvor det er detektert mer enn seks QR-koder. Resultatene er presentert slik at standardavviket er på y-aksen, og x-aksen viser hvor stor del av målesettet som innfrir betingelsen som er satt. Resultater lengst mulig nede i det høyre hjørnet av plottet blir da mest ønskelig.

Testbilde 1

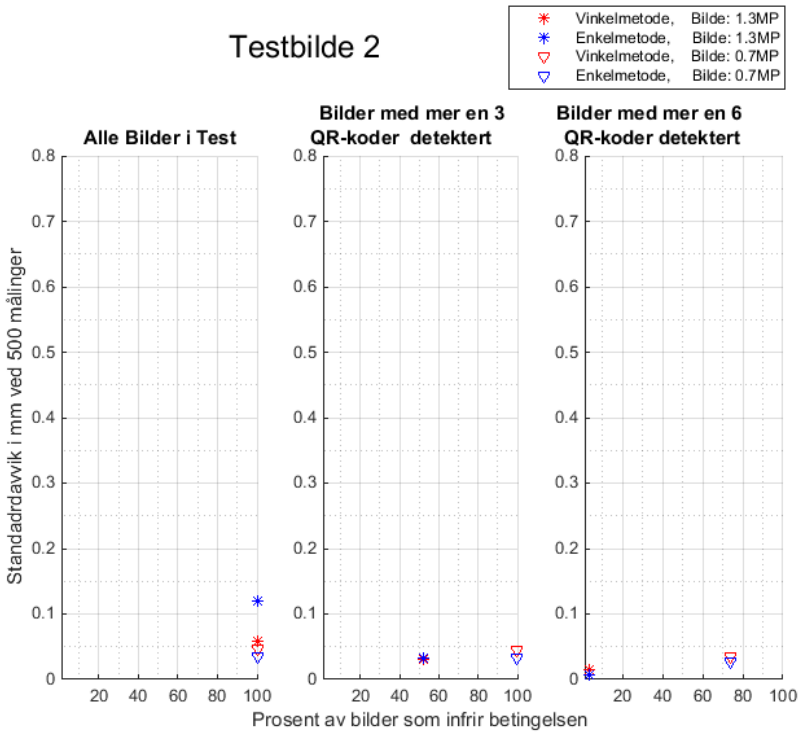
Testbilde 1 er tatt slik at flest mulig QR-koder er med i bildet. Figur 53 viser bildet. Resultatene er presentert i Figur 54. Det oppnåelige standardavviket er på 0,1 mm når alle bilder er med i beregningen. Ved å begrense målegrunnlaget til de bilder med mer enn seks detekterte QR-koder, går standardavviket ned mot 0,05 mm. Spesielt Enkelmetode har mye forbedring ved å begrense målingene slik. Ved denne begrensingen blir det stor økning i antall godkjente målinger ved å redusere oppløsningen.



Figur 53. Testbilde 1. Bildet er tatt slik at flest mulig QR-koder kan bli detektert.



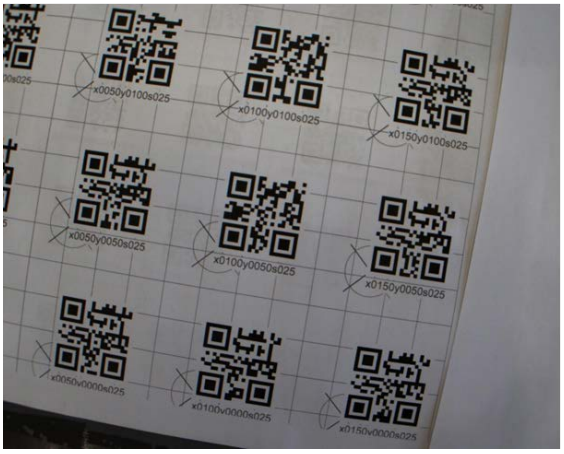
Figur 54. Resultater fra testbilde 1. Y-aksen viser målt standardavvik, og x-aksen viser hvor stor del av målingene som infrir betingelsen. Resultater lengst nede til høgre er det mest ønskelige. Standardavviket ligger rundt 0,1 mm for de fleste estimatene, men går opp mot 0,7 mm for Enkelmetode. Det er svært få av målingene fra 1,3 MP som finner mer enn 6 QR-koder.



Figur 55. Resultater for testbilde 2. Y-aksen viser målt standardavvik, og x-aksen viser hvor stor del av målingene som infrir betingelsen. Resultater lengst nede til høgre er det mest ønskelige. Standardavviket ligger rundt 0,05 mm for de fleste estimatene. Bare 50% av målingene for 1,3 MP finner mer enn 3 QR-koder, og det er svært få målinger som finner mer enn 6 QR-koder.

Testbilde 2

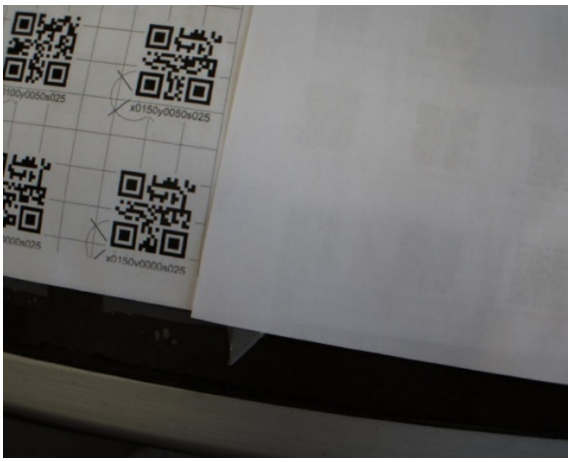
Testbilde 2 er tatt slik at litt færre QR-koder synes enn i testbilde 1, men her er en av QR-kodene plassert midt i bildet. Det er da fire hjørnepunkter som omkranser senterpunktet, og det gir et godt beregningsgrunnlag. Bildet kan sees i Figur 56. Resultatene er presentert i Figur 55. Her er det generelt bedre resultater enn i Testbilde 1. De oppnåelige standardavviket er svært likt både med og uten begrensingene, og ligger i underkant av 0,05 mm. Igjen er det Enkelmetode som har mest forbedring ved flere QR-koder. Også her er delen av godkjente målinger mye større ved redusert oppløsning.



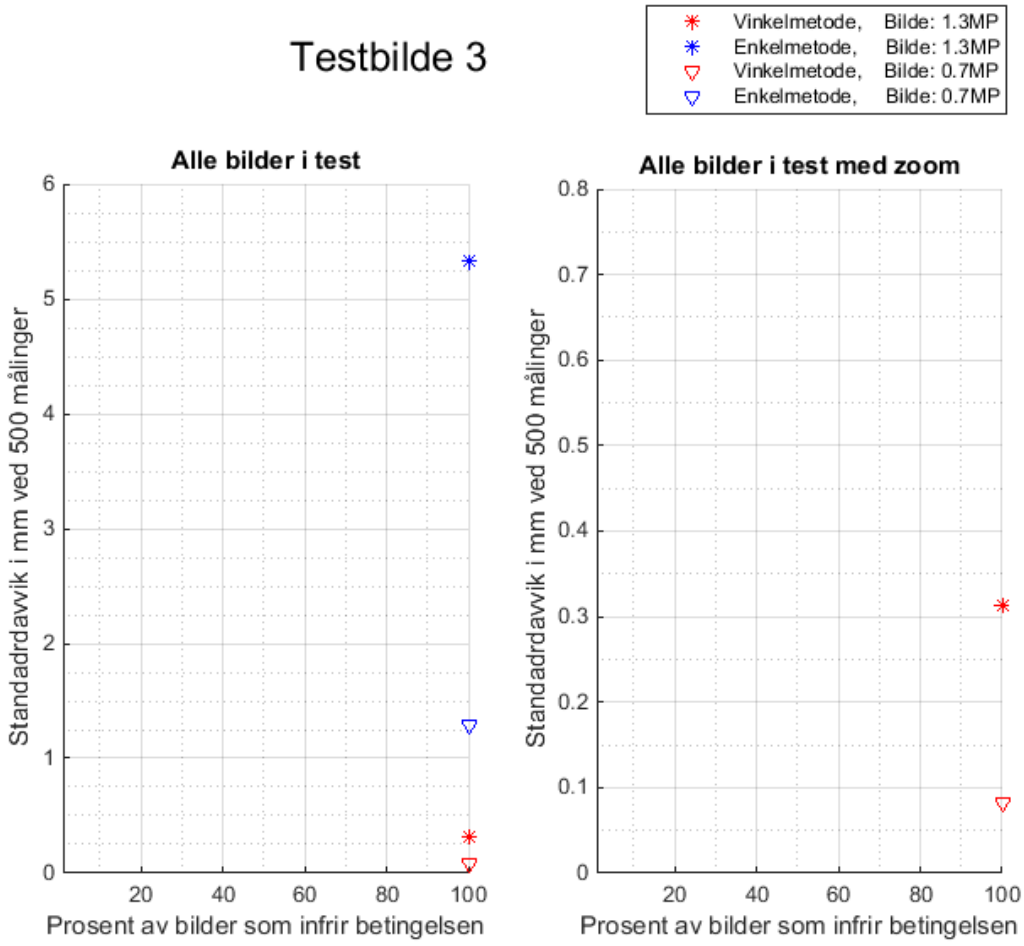
Figur 56. Testbilde 2. Bildesenteret er godt plassert midt i en QR-kode.

Testbilde 3

Testbilde 3 kan sees i Figur 57. Her er det bare synlige QR-koder i et av hjørnene. Det er ikke mulig å detektere mer enn tre QR-koder. Det har derfor ingenting for seg å se på de betingelsene som er gjort for testbilde 1 og 2. Figur 58 viser resultatene i en versjon der alle verdiene er synlige, og en innzoomet versjon som er skalert lik som resultatene for testbilde 1 og 2. Her er svakheten ved Enkelmetode tydelig med et standardavvik på over 5 mm når det blir få QR-koder tilgjengelig. Vinkelmetode viser derimot fortsatt et standardavvik på under 0,1 mm når oppløsningen er 0,7 MP.



Figur 57. Testbilde 3. Få QR-koder kan bli detektert.

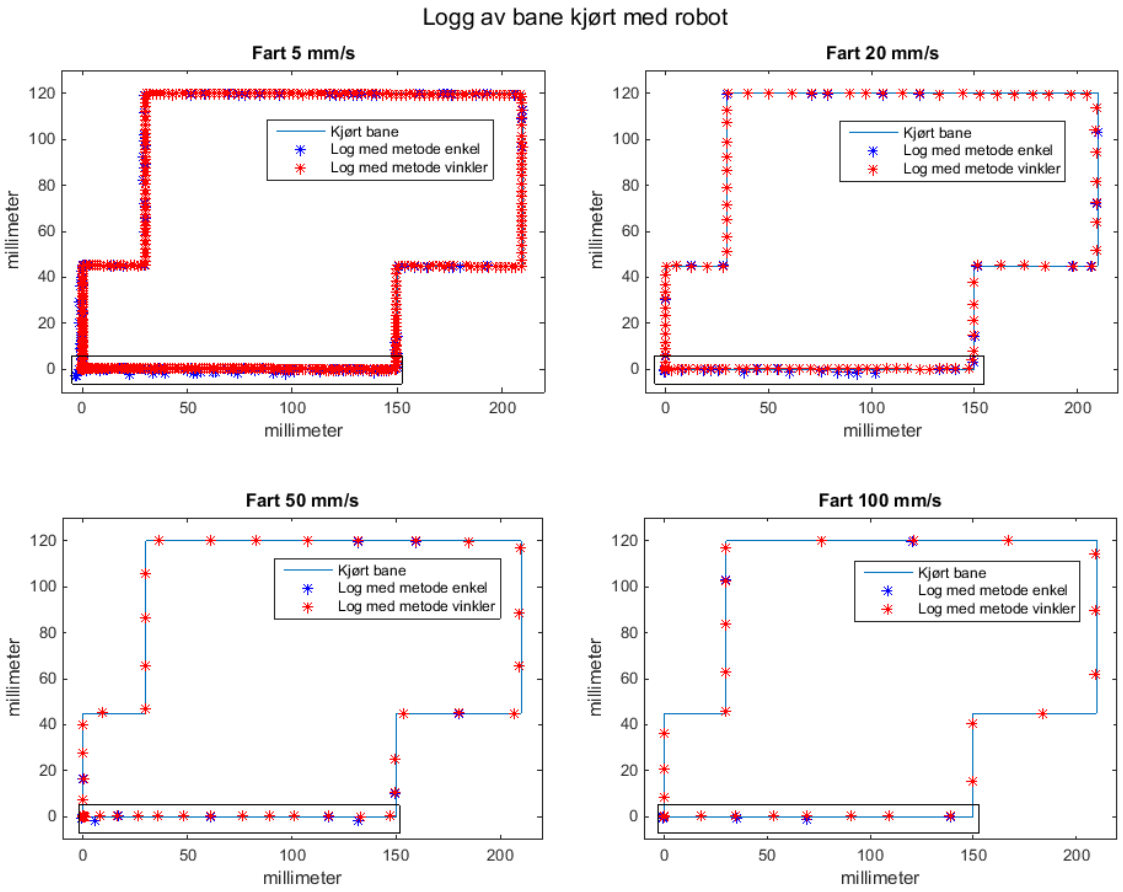


Figur 58. Resultater fra testbilde 3. Y-aksen viser målt standardavvik. Standardavviket for Vinkelmetode 0,7 MP ligger i det samme området som Testbilde 1 og 2 . For 1,3 MP og Enkelmetode er standardavvikene langt større for dette bildet.

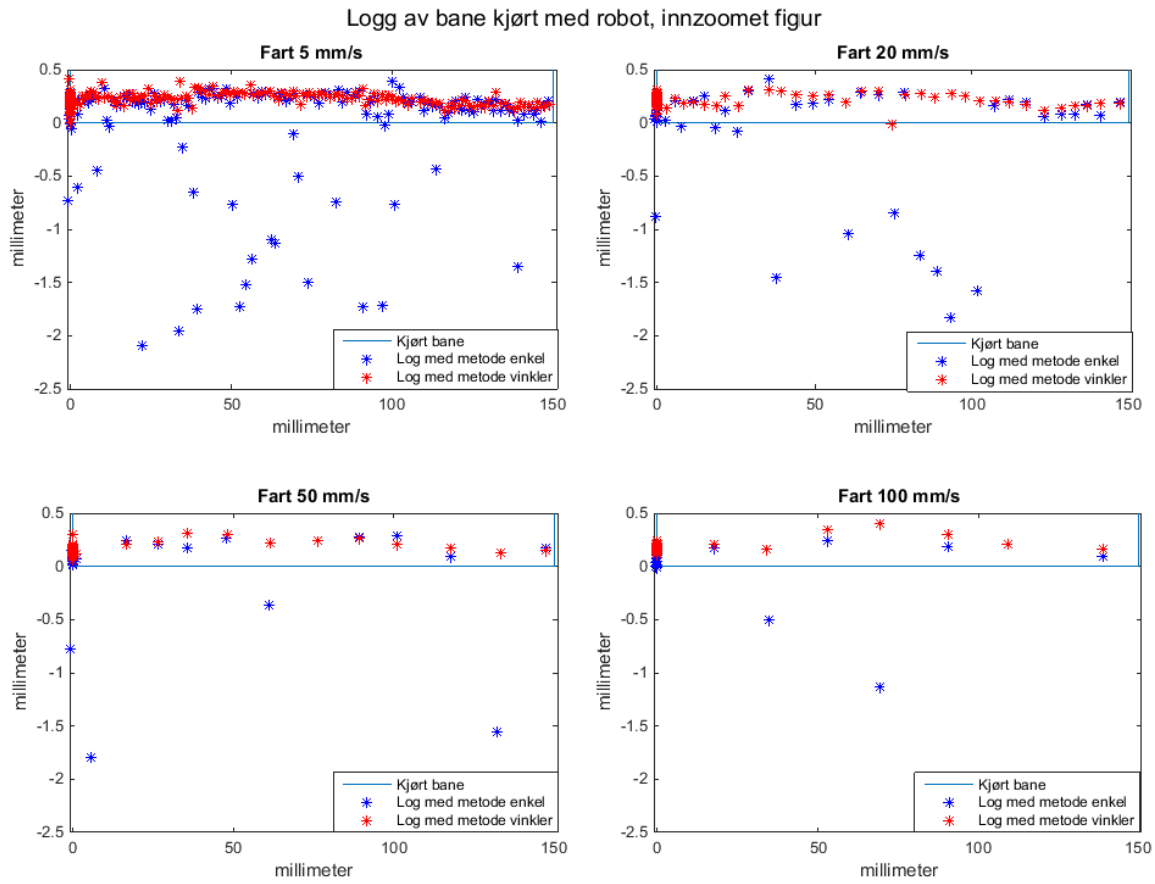
4.10 Test av posisjonsestimat med kamera i bevegelse

Denne testen vurderer posisjonsestimater når kameraet beveger seg over QR-matrisen. Testoppsett 2 fra kap. 4.1 er brukt. Roboten som holder kameraet får kommando om å gå i en forhåndsprogrammert bane over QR-matrisen. Banen kan sees sammen med resultatene i Figur 59. Farten til roboten kan justeres, og denne testen er kjørt flere ganger med fart fra 5 til 100 mm/s.

Resultatene i Figur 59 viser fire relevante kjøringer med fart på 5, 20, 50 og 100 mm/s. Figur 60 viser de samme resultatene, men her er det zoomet inn på et lite område for å se nøyaktigheten i estimatene. I testen er både Enkelmetode og Vinkelmetode brukt for å estimere posisjon.



Figur 59. Resultatene fra test av systemet i bevegelse. Den blå stripen viser den banen som roboten følger. De blå stjernene viser estimatene fra Enkelmetode og de røde fra Vinkelmetode. Den sorte firkanten indikerer området som er zoomet inn på i Figur 60. Resultatene viser at posisjonsestimatet følger banen slik som ønsket.



Figur 60. Dette er et utnsnitt av den sorte firkanten i Figur 59. Her er estimatene fra Enkelmetode veldig spredd i forhold til Vinkelmetode. Presisjonen for Vinkelmetode er innenfor det som er forventet fra kap. 4.9. Det er også et fast avvik på ca. 0,25 mm for estimatene.

Estimatene følger banen under alle de utprøvde hastighetene. Presisjonen til Enkelmetode varierer innenfor et område på 2,5 mm. For Vinkelmetode varierer estimatene innenfor område på 0,3 til 0,4 mm. Presisjonen endre seg ikke vesentlig mellom de ulike hastighetene. Posisjonsestimatene har et fast avvik på ca. 0,25 mm.

5. Diskusjon

Diskusjonskapittelet vurderer resultatene fra forsøkene som er gjort i kapittel 4 og gjør valg av algoritmer og parametere basert på dette. Det blir også diskutert rundt hva som kan forårsake forskjeller for de ulike algoritmene. Videre er det satt opp en anbefaling av systemoppsettet. Det blir til slutt opplyst om hvilke ting som ikke er prioritert i testene og hva som kan jobbes videre med.

5.1 Valg av optimale algoritmer til systemet

Algoritmen Snitt av linjer velges for å estimere rotasjon rundt z-aksen, og algoritmen Kort linje vertikalt velges for å estimere pikselstørrelsen. For å estimere senterposisjonen til bildet velges algoritmen Vinkelmetode, men Enkelmetode anbefales om det er stort antall QR-koder tilgjengelig. Disse valgene er basert på testene i kap. 4.2, 4.3 og 4.4, og videre i dette kapittelet er grunnlaget for valgene diskutert.

Valg av algoritme for å fastsette rotasjon om z-aksen

Testen av algoritmer for å fastsette rotasjon rundt z-aksen i kap. 4.2 viser ganske klart at algoritmen Snitt av linjer gir det beste estimatet av rotasjonen. Dette viser seg spesielt i resultatene fra de bildene som ikke er korrigert for linseforvrengning. Toleransen for linseforvrengning har trolig sammenheng med at denne algoritmen er en funksjon av alle QR-kodene i bildet. Linseforvrengning påvirker omvendtproporsjonalt over og under samt på venstre og høre side av midten til bildet. Feil som skyldes linseforvrengning vil da oppstå med motsatt fortegn på to ulike sider i bildet, og de vil i stor grad kansellere hverandre når punkter fra store deler av bildet blir brukt i et gjennomsnitt.

Snitt av linjer gir en målt feil på 0,0° med presisjon på ±0,05°. Denne algoritmen har lengst prosesstid av de som er testet, ca. 2,5 ganger lenger enn de konkurrerende algoritmene. En oversikt over testresultatene fra kap. 4.2 er presentert i Tabell 4. Om det skal lages et system der behandlingstid er essensiell, kan algoritmen Tilpasset linje vertikalt velges istedenfor Snitt av linjer. Det er i så fall nødvendig å bruke linsekorreksjon. Feilen for denne algoritmen er målt til 0,1°.

Tabell 4. Oversikt over målefeil og prosesstid fra testene av de ulike algoritmene for å fastsette rotasjon rundt z-aksen. Tallene er hentet fra resultatene i kap. 4.2.

Algoritme for å finne rotasjon rundt z-aksen	**Målt feil med linsekorreksjon**	**Målt feil uten linsekorreksjon**	**Prosesstid**
Lang linje	0,2°	0,2°	15 ms
Kort linje	0,3°	0,3°	15 ms
Tilpasset linje horisontalt	0,2°	0,3°	14 ms
Tilpasset linje vertikalt	0,1°	0,5°	14 ms
Snitt av linjer	0,0°	0,0°	35 ms

Valg av algoritme for å fastsette pikselstørrelsen i romkoordinater

Testen av algoritmene som skal fastsette pikselstørrelsen i kap. 4.3 viser entydig at algoritmen Kort linje vertikalt gir det beste resultatet både med og uten linsekorreksjon. Det er målt 0 µm feil med presisjon på ±0,6 µm når linsekorreksjon er brukt, og 1 µm feil uten linsekorreksjon. Tabell 5 viser en oversikt over resultatene fra testen.

Tabell 5. Oversikt over målefeil og prosesstid fra testene av de ulike algoritmene for å fastsette pikselstørrelsen. Tallene er hentet fra resultatene i kap. 4.3.

Algoritme for å finne pikselstørrelsen	**Målt feil med linsekorreksjon**	**Målt feil uten linsekorreksjon**	**Prosesstid**
Lang linje horisontalt	5 µm	3 µm	3 ms
Lang linje vertikalt	1 µm	1 µm	3 ms
Kort linje horisontalt	0 µm	2 µm	3 ms
Kort linje vertikalt	0 µm	1 µm	3 ms
Lengst mulig linje	7 µm	3 µm	12 ms
Snitt av linjer	6 µm	6 µm	6 ms
Snitt av linjer med QR-koder nærmere enn 600 piksler fra senter	5 µm	3 µm	6 ms

Noen av de andre testene i kap. 4 har vist antydninger til at pikselverdien kan ha noe mer usikkerhet enn hva som har vist seg i den spesifikke testen av pikselstørrelse. Vinkelmetode er den algoritmen som fungerer best for posisjonsestimat, og denne algoritmen bruker ikke pikselstørrelse som parameter. Radiusmetode bruker ikke vinkler, men kun pikselstørrelse som parameter, og dette er den algoritmen som viser dårligst resultat for posisjonsestimat. Det er derfor trolig en feil eller usikkerhet i estimatene av pikselstørrelse som ikke har kommet godt nok frem.

Fasitverdien til pikselstørrelsen var målt i senter av bildet, se kap. 4.1. Det er sannsynlig at algoritmen Kort linje vertikalt fungerte best nettopp fordi den bruker verdier bare fra det området.

En mulig grunn til at pikselstørrelsen ikke alltid stemmer like godt er at pikslene kan ha ulik størrelse i ulike deler av bildet. Perspektivforvrengning som er beskrevet i teorikapittel 2.2.3, vil føre direkte til et slikt resultat, men den bør teoretisk sett ikke gjøre seg gjeldene når motivet er plant og kameraet står normalt mot motivet. Linseforvrengning som er beskrevet i teorikapittel 2.2.2, vil også kunne føre til ulike pikselstørrelser i bildet. Dette burde i teorien bli løst med linsekorreksjon, men selv med korrigering er det ikke sikkert at alle pikslene blir 100% riktige.

Kort linje vertikalt ble valgt som algoritme for å fastsette pikselstørrelsen der det var behov for den i testene videre i kap. 4. Den anbefales fortsatt om Enkelmetode skal brukes for å estimere posisjonen. Om Vinkelmetode benyttes, er ikke pikselstørrelsen nødvendig å fastsette.

Valg av algoritme for å estimere senterpunktet til bildet i romkoordinater

En av de tre algoritmene, som er laget i dette prosjektet, for å estimere senter av bildet i romkoordinater utmerker seg negativt. Radiusmetode har vist dårligere resultater enn de to konkurrerende algoritmene i alle situasjoner. Denne algoritmen baserer seg kun på pikselstørrelsen og ikke rotasjon rundt z-akse. Pikselstørrelsen har mest trolig dårligere presisjon enn rotasjonen, dette er forklart tidligere i kapittelet.

En oversikt over resultatene fra testen i kap. 4.4 er presentert i Tabell 6. Enkelmetode baserer seg på både pikselstørrelse og rotasjon rundt z-aksen. Denne algoritmen har svært like resultater som Radiusmetode, men de er i alle tilfeller bedre, Figur 41 viser dette godt. Ved stort antall QR-koder, har Enkelmetode i noen tilfeller bedre resultat enn Vinkelmetode. Ett gjennomsnitt av mange QR-koder ser ut til å oppheve de feilene som hver QR-kode har. Dette indikerer at feilene kan komme fra kilder som er speilet om midten av bildet. Sannsynligvis er det pikselstørrelsen som blir større og større desto lenger mot bildekanten pikslene ligger. Dette vil resultere i at den estimerte feilen for hver QR-kode blir større og større desto lenger de ligger fra senter av bildet. Dette er noe som testene i kap. 4.3.1 og 4.4.1 også indikerer. QR-kodene på venstre og høyre side, samt over og under midten, vil få feil med motsatt fortegn av hverandre. Med jevnt fordelte QR-koder i bildet vil de kunne oppheve hverandre.

Vinkelmetode har svært mye bedre presisjon ved få QR-koder enn hva de to andre algoritmene har. Denne algoritmen virker mye mer robust enn de andre. Dette har trolig sammenheng med at den ikke bruker pikselstørrelse som parameter. Vinkelmetode er den generelt beste algoritmen og anbefales til videre bruk. Enkelmetode kan med fordel bruks om det er detektert mange, gjerne over 10, QR-koder i bildet.

Tabell 6. Presisjon og prosesstid for algoritmene som estimerer senterpunktet til bildet i romkoordinater. Verdiene er hentet fra testene i kap. 4.4. De har som mål å sammenligne algoritmene, og presisjonen i tabellen er derfor ikke optimalisert slik som den er i test av presisjon i kap. 4.9. Optimal presisjon er presentert i Tabell 7.

Algoritme for å estimere senterpunktet til bildet i romkoordinater	**Presisjon med over 8 QR-koder**	**Presisjon med 4 til 8 QR-koder**	**Presisjon med 1 til 4 QR-koder**	**Prosesstid**
Vinkelmetode	< 0,2 mm	< 0,3 mm	0,3-0,75 mm	10 ms
Enkelmetode	< 0,2 mm	0,4-5 mm	5-10 mm	10 ms
Radiusmetode	0,3-0,8	0,8-10 mm	10-20 mm	6 ms
EPnP	0,75-4 mm	0,75-4 mm	0,75-4 mm	3 ms

EPnP er en ekstern algoritme som har vært testet sammen med de andre tre algoritmene i kap. 4.4. Den viste svært mange tilfeller hvor estimatet overgikk alle grenser. Figur 40 viser dette. Ved å luke ut disse feilene, ble algoritmen likevel sammenlignet med de tre andre. Den har bedre presisjon enn både Radiusmetode og Enkelmetode når det er få QR-koder detektert. Vinkelmetode fungerer derimot i alle tilfeller bedre enn EPnP.

Det er vanskelig å si noe om grunnen til disse hyppige feilene i EPnP. Denne algoritmen er kjørt direkte i OpenCV og får bare QR-kodenes punkter som input, ikke pikselstørrelse eller rotasjon. Algoritmen er beskrevet i *EPnP: An Accurate O(n) Solution to the PnP Problem* [25]. Her vises det til en ustabilitet i algoritmen i de tilfellene hvor alle punktene ligger i det samme planet. Det er tilfelle i testoppsettet i dette prosjektet, men det er også påstått at ustabiliteten ikke oppstår hvis kameraet står normalt på planet [25]. Det må regnes som sannsynlig at det er en sammenheng med denne ustabiliteten selv om kriteriet om kamera normalt på planet er oppfylt i testene.

5.2 Valg av parametere i systemoppsettet

Valg av oppløsning på kamera

Testen av oppløsning i kap. 4.7 vise at oppløsningen ikke må være for lav. Ved oppløsning på 0,3 megapiksler(MP) og 0,2 MP, er det svært få QR-koder som blir detektert. Oppløsning rundt 0,7 MP ser ut til å fungere best. Ved å øke oppløsningen til 1,3 MP, er det færre QR-koder som blir detektert. Presisjonen til posisjonsestimatet blir bedre med høyere oppløsning forutsatt like mange QR-koder detektert, men antall QR-koder har mer å si for presisjonene enn hva oppløsningen har når Vinkelmetode brukes. Figur 48 og Figur 49 viser dette.

Biblioteket ZBar, som skanner bildet for QR-koder, ser på rådata av bildet og behandler intensitetsnivået mellom pikslene for å avgjøre hvor kantene til QR-koden er. Dokumentasjonen til dette biblioteket er dessverre svært mangelfull, og det gjør det vanskelig å forstå detaljene i virkemåten. En høyere oppløsning vil gi en ikke fullt så skarp overgang i kantene på QR-kodene, og det kan være medvirkende til at færre QR-koder blir detektert av ZBar.

Skal systemet brukes online, anbefales en oppløsning på ca. 0,7 megapiksler. Skal derimot systemet brukes til analyse av lagrede bilder, eller i trege systemer der tid ikke spiller så stor rolle, kan bildene med fordel skannes på flere oppløsninger og velges etter hvor mange QR-koder som blir funnet.

Valg av størrelse på QR-koder

Testen i kap. 4.5 viser at QR-koder med størrelse mellom 20x20 mm og 25x25 mm gir flest oppdagede QR-koder. Disse størrelsene er relative til både objektivtype og avstand til motivet. For lettere å kunne adaptere systemet til andre avstander er det videre i dette kapittelet utviklet noen formler for å regne ut hvilken størrelse QR-kodene må ha som funksjon av avstand til motivet og type objektiv.

For å finne synsfeltet til kameraet i millimeter, er det mulig å ta bilde av rutenettet til QR-matrisen, en linjal eller lignende. Det kan også finnes med litt informasjon fra datablad til kamera og objektiv som i teorien i kap. 2.2.1.

Det er ønskelig å finne H_r og W_r som henholdsvis er høyden og bredden til synsfeltet oppgitt i millimeter. H_b og W_b er høyden og bredden i bildet oppgitt i piksler. I tillegg til høyden og bredden, trengs p, som er pikselstørrelsen på bildebrikken og f, som er brennvidden til objektivet. Alt dette kan hentes fra databladene til kamera og objektiv. Ved å sette disse parameterne i ligning (1) og (2) fra kap. 2.2.1, blir størrelsen på synsfeltet som ligning (26) og (27) viser. Parameter d er avstanden til motivet, og den er her satt til 0,2 m slik som den var under testen i kap. 4.5.

Pikselstørrelsen p som vises til her er den fysiske størrelsen til en piksel på bildebrikken, og må ikke forveksles med pikslenes størrelse i romkoordinater slik som i kap. 3 og 4.

$$H_r = \frac{p \cdot H_b \cdot d}{f} = \frac{5{,}3 \cdot 10^{-6} \cdot 1024 \cdot 0{,}2}{8 \cdot 10^{-3}} = 0{,}136m = 136mm \quad (26)$$

$$W_r = \frac{p \cdot w_b \cdot d}{f} = \frac{5{,}3 \cdot 10^{-6} \cdot 1280 \cdot 0{,}2}{8 \cdot 10^{-3}} = 0{,}170m = 170mm \quad (27)$$

Av arealet til synsfeltet utgjør en QR-kode:

$$prosentdel = \frac{\begin{bmatrix} min \\ max \end{bmatrix}^2}{H_r \cdot w_r} \cdot 100\% = \frac{\begin{bmatrix} 20mm \\ 25mm \end{bmatrix}^2}{136mm \cdot 170mm} \cdot 100\% = \begin{bmatrix} 1{,}7\% \\ 2{,}7\% \end{bmatrix} \quad (28)$$

En QR-kode bør utgjøre mellom 1,7% og 2,7 % av arealet til kameraets synsfelt. Formel (28) kan lett snues om på, slik som (29), for å finne størrelsen til en QR-kode fra et vilkårlig synsfelt. Enda mer kompakt kan det skrives som (30). Basert på oppløsning, brennvidde, pikselstørrelse og avstand til motivet, finner formel (30) maksimalt og minimal størrelse på QR-kode. Millimeter eller meter kan brukes som enhet etter det som er ønsket såfremt alle mål er i samme enhet. Merk att oppløsningen er bildebrikken sin fysiske høyde og bredde, og den må ikke forveksles med bildets oppløsning når den blir redusert slik som i kap. 4.7.

$$\begin{bmatrix} min \\ max \end{bmatrix} = \sqrt{\frac{\begin{bmatrix} 1{,}7\% \\ 2{,}7\% \end{bmatrix} \cdot H_r \cdot w_r}{100\%}} = \sqrt{\begin{bmatrix} 0{,}017 \\ 0{,}027 \end{bmatrix} \cdot H_r \cdot w_r} \quad (29)$$

$$\begin{bmatrix} min \\ max \end{bmatrix} = \frac{p \cdot d}{f} \cdot \sqrt{\begin{bmatrix} 0{,}017 \\ 0{,}027 \end{bmatrix} \cdot H_b \cdot w_b} \quad (30)$$

Faren ved å bruke QR-koder som er mindre enn de som anbefales her er at få QR-koder blir detektert. Faren med å bruke for store QR-koder er at det blir færre QR-koder å estimere posisjonen fra. Skal målingene brukes i et system hvor effektivitet er prioritert før nøyaktighet, kan QR-størrelsen med fordel økes opp til mellom tre og seks prosent av bildet. Det resulterer i raskere prosesstid, men litt redusert presisjon.

Valg av objektiv og bruk av linsekorreksjon

Testene i kap. 4.6 viser at ulike brennvidder på objektivet ikke gav noen vesentlige forskjeller på resultatet av posisjonsestimatet. Det har ikke latt seg gjøre å anskaffe teleobjektiv til kameraet, og det er derfor bare testet brennvidder i vidvinkelområdet. Den minste brennvidden som er testet er 6 mm, og den har da et avvik på 3 mm fra normalobjektivet som har 9 mm brennvidde. Testene er gjort på en slik måte at det bare er endring av brennvidde, og ikke motivet, som skal ha påvirkning på estimatet. Et teleobjektiv med 3 mm avvik vil trolig gi en forvrenging i samme størrelsesorden, men i motsatt retning, se kap. 2.2.2 om linseforvrengning. Det blir på grunnlag av dette gjort en antagelse om at et teleobjektiv heller ikke vil gi vesentlige feil. Det settes derfor ingen begrensninger på hvilke objektiver som kan brukes med systemet.

Bruk av linsekorreksjon avhenger av at kameraet og objektivet sine interne parametere er funnet. For å finne disse kreves en kamerakalibreringsrutine på forhånd, slik som forklart i kap. 2.2.4.

Testene av påvirkning fra linsekorreksjon i kap. 4.6 viser at når Vinkelmetode blir brukt, er forbedringen minimal i de fleste tilfeller. Det er en forbedring opp mot 30% når det er detektert mange QR-koder i hele bildet, og i de tilfellene er nøyaktigheten allerede svært høy. Dette viser at ved bruk av denne algoritmen kan kamerakalibrering og linsekorreksjon sløyfes om ønskelig. Det blir da raskere og enklere å sette opp et system. Om flere objektiver eller kameraer skal testes på et system, kan de faktisk bare kobles til og kjøres uten at kalibrering og innstillinger er nødvendig for å få en god presisjon.

Ved bruk av Enkelmetode er det mer forskjell på resultatene med og uten linsekorreksjon. Her er forbedringen på mellom 30 og 85%. Dette har trolig en sammenheng med at algoritmen gjør bruk av pikselstørrelse som har blitt omtalt tidligere i kap. 5.1. Pikselstørrelsen har trolig en varierende størrelse i bildet, og dette blir korrigert noe med linsekorreksjon. I kap. 5.1 ble Enkelmetode anbefalt til bruk når det er detektert mange QR-koder, og ikke ellers. Fordelen til denne algoritmen er ikke gjeldene uten linsekorreksjon, og gjør derfor at algoritmen blir overflødig om ikke linsekorreksjon brukes.

Det er ulike måter å bruke linsekorreksjon på. For å korrigere et bilde slik at det ser naturlig ut for øye, må hele bildet korrigeres. Forskjellen mellom et korrigert og et ikke korrigert bilde er vist i Figur 26 og Figur 27. For systemet i dette prosjektet er det derimot unødvendig å korrigere hele bildet på denne måten. Det eneste systemet bruker av et bilde etter det er skannet av ZBar er hjørnekoordinatene til hver QR-kode, og det holder da å korrigere disse punktene. Forskjellen i behandlingstid er enorm. I et 1 megapikselbilde der det er oppdaget ti QR-koder, er det snakk om å behandle 40 piksler istedenfor en million.

Vinkelen til kameraet mot Qr-matrisen

Prosjektet har siden starten hatt en betingelse om at kameraet skal stå normalt mot QR-matrisen. Dette ble utfordret i kapittel 4.8 for å se om systemet kan brukes utover betingelsen. Testen viser at systemet, med bruk av Vinkelmetode, er upåvirket av vinkel opp til 5°. Med vinkler opp mot 15°, fungerer estimatet fortsatt godt, men presisjonen er redusert til ±1 mm. Med vinkler utover dette, øker feilene mye. Det anbefales ikke å bruke vinkel over 15°, og for presise systemer anbefales ikke vinkel over 5°.

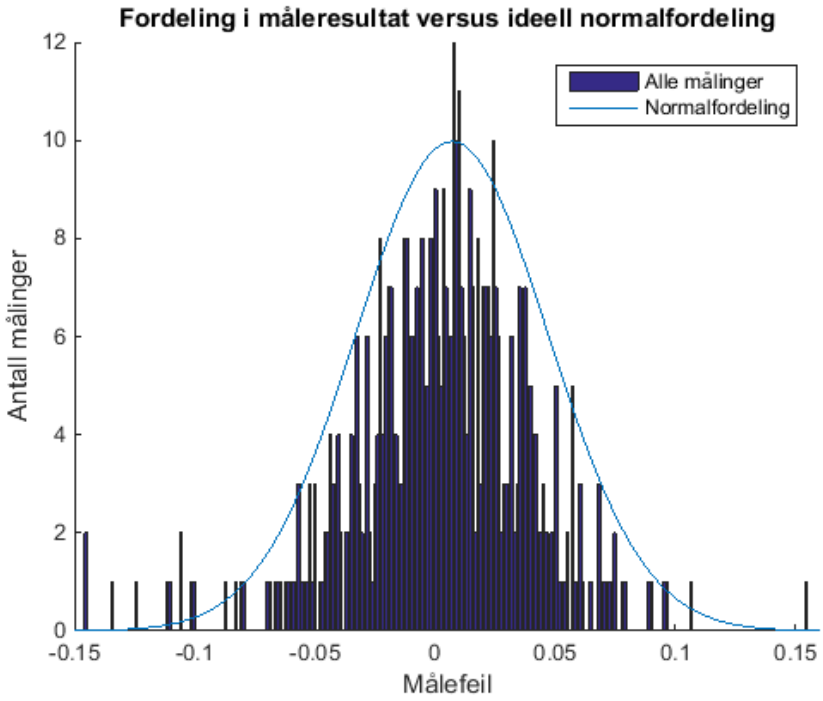
Det er to ting som påvirker estimatene ved vinklet kamera. Den første og mest direkte konsekvensen er perspektivforvrengning, kap. 2.2.3. Med vinkel på kameraet følger det at avstanden til QR-kodene blir ulik. Dette fører igjen til at QR-kodene i bildet ikke har den synkrone plasseringen og heller ikke den kvadratiske formen som de har i virkeligheten. Dette fører spesielt til at pikselstørrelsen blir ulik i bildet. Rotasjonen kan også bli påvirket av dette. I tillegg er ikke hjørnepunktene til QR-kodene symmetrisk plassert i bildet slik som de er på QR-matrisen.

Den andre påvirkningen er fokusplanet. Problemet oppstår spesielt med vinkler over 15° der avstanden til delene av motivet blir så forskjellige at det er vanskelig å få godt nok fokus til å tolke QR-koden over hele bildet. Dette er tydelig til venstre i Figur 5.

Begge disse utfordringene vil i teorien kunne begrenses noe ved å bruke ett teleobjektiv med lenger avstand til QR-matrisen.

5.3 Vurdering av presisjonen til systemet

Presisjonen til systemet er testet i kap. 4.9. Presisjonen er gitt som ± tre standardavvik (±3σ) fra 500 estimater av posisjonen til senter av bildet. Dette er basert på at systemets posisjonsestimater er normalfordelte. For å underbygge denne påstanden, er alle estimatene for ett av testbildene fra kap. 4.9 plottet i Figur 61. I samme plott er også en ideell normalfordeling basert på standardavviket plottet, og samsvaret er stort. For å sikre at eventuelle outliers blir oppdaget, er også maksimalt avvik lagret. Maksimalt oppdagede avvik var i alle testene i området under 3.75σ, noe som også er rimelig i et normalfordelt system.



Figur 61. Alle målinger gjort med Vinkelmetode på testbilde 3 ved 0,7 megapiksler. Sammenlignet med normalfordelingen med σ lik 0,04 mm, underbygger dette påstanden om at målingene er normalfordelte.

Tabell 7. Oversikt over resultatene fra test av presisjon i kap. 4.9. Verdiene er oppgitt som ±3σ der minste verdi er når flest QR-koder i testen er detektert og høyeste når færrest QR-koder er detektert. Verdiene for σ er hentet fra Figur 54, Figur 55 og Figur 58.

Algoritme	Enkelmetode		Vinkelmetode	
Oppløsning	1,3 MP	0,7 MP	1,3 MP	0,7 MP
Testbilde 1, 1 -12 QR-koder	0,1 - 2 mm	0,25 – 0,6 mm	0,1 - 0,35 mm	0,2 – 0,3 mm
Testbilde 2, 1-9 QR-koder	0,1 - 0,35 mm	<0,1 mm	0,1 - 0,15 mm	<0,1 mm
Testbilde 3, 1-3 QR-koder	15 mm	3,5 mm	1 mm	0,25 mm

Generelt for de tre testbildene som er testet i kap. 4.9 er Vinkelmetode mest robust. Den viser en feil som aldri overstiger ±0,3 mm, og med mange QR-koder er den under ± 0,1 mm. Enkelmetode fungerer dårlig på få QR-koder sammenlignet med Vinkelmetode, men ved å sette krav til antall QR-koder, blir presisjonen til Enkelmetode like god og i noen tilfeller bedre enn Vinkelmetode. En grov oversikt over resultatene fra kap. 4.9 er presentert i Tabell 7.

Når mange QR-koder er detektert, viser resultatene bedre presisjon for 1,3 megapiksler (MP) enn for 0,7 MP, men den delen av målingene som har mange QR-koder detektert er svært liten for 1,3 MP i forhold til 0,7 MP. Det er heller ikke problemfritt å sammenligne standardavviket i de tilfellene hvor målesettet blir så lite som for eksempel for 1,3 MP til høyre i Figur 55. Her er det under 5% av målingene som innfrir betingelsen om seks QR-koder. Det betyr under 25 målinger, og det lave standardavvik kan like gjerne komme av tilfeldigheter.

Posisjonsestimatet med bruk av Vinkelmetode har et standardavvik som alltid er under 0,1 mm, og med mange QR-koder kommer det ned mot 0,03 mm. Enkelmetode med krav til mange QR-koder, gjerne over 10, gir standardavvik som er enda lavere. Med utgangspunkt i normalfordelte målinger, tilsvarer dette en presisjon på under ±0,3 mm.

Resultatene som er nevnt her er relative til pikselstørrelsen, eller hvor stor avstand i romkoordinatsystemet en piksel representerer. For eksempel vil et system med en QR-matrise på 200 meters hold der en piksel representerer 20 cm av motivet, aldri oppnå den presisjonen som er nevnt her. Det er derfor nødvendig å representere presisjonen som funksjon av pikselstørrelsen. Pikselstørrelsen er proporsjonal til avstanden mellom kamera og motiv. Å oppgi presisjonen som en funksjon av denne avstanden ville være fordelaktig fordi det gjerne er den parameteren som er lettest å finne. Dette forholdet er på den andre siden relatert til objektivets brennvidde, og vil derfor ikke være generelt nok. Presisjonen blir derfor oppgitt som en funksjon av pikseltørrelsen. Pikselstørrelsen i et bilde kan enkelt finnes ved hjelp av algoritmene fra kap. 3.2.

I testene med avstand på 200 mm mellom kamera og motiv, og med brennvidde på 8 mm, er pikselstørrelsen på 0,15 mm. Presisjonen som tidligere har vert oppgitt som under ±0,3 mm, blir med formel (31) under ±2 ganger pikselstørrelsen.

$$Generell\ presisjon = \frac{spesifikk\ presisjon}{spesifikk\ pikselstørrelse} = \frac{0{,}3mm}{0{,}15mm} = 2 \tag{31}$$

Presisjonen til systemet i bevegelse

Testen av kamera i bevegelse over QR-matrisen fra kap. 4.10 gjenspeiler resultatene fra presisjon i kap. 4.9. Enkelmetode har svært stor spredning i posisjonsestimatet sammenlignet med Vinkelmetode. I denne testen er det ingen kontroll på hvor mange QR-koder som er oppdaget. Posisjonen er estimert med begge algoritmer på grunnlag av de QR-kodene som er tilgjengelig. Figur 60 viser et utsnitt av den estimerte banen. Her er det en maksimal spredning for estimatene fra Vinkelmetode i et område på ca. 0,4 mm, og dette er innenfor presisjonen på ±0,3 mm som er anslått fra den stillestående presisjonstesten.

Fra Figur 59 er det tydelig at oppdateringsfrekvensen gjør at estimatene ikke kommer ofte nok til å estimere banen presis ved høy fart. Programmet som kjører algoritmene er ikke optimalisert for rask prosessering og i tillegg estimeres posisjonen to ganger, en gang for hver algoritme. Oppdateringsfrekvensen er på ca. 4Hz. Det er likevel viktig å legge merke til at estimatene er like presise i høy hastighet.

Posisjonsestimatet har et fast avvik på ca. 0,25 mm, og det er spesielt tydelig for Vinkelmetode i Figur 60. Dette skyldes trolig en liten feil i synkronisering mellom roboten og QR-matrisen. Dette underbygger også bruk av standardavvik som måleparameter istedenfor gjennomsnittlig posisjon som forklart i starten av kap. 4.

En film av test i bevegelse med fart på 30 mm/s og posisjonsestimat fra Vinkelmetode kan sees ved å følge linken: https://www.youtube.com/watch?v=32ao60naKok. Her er det tydelig at oppdateringsfrekvensen blir litt lav i forhold til farten, spesielt når det er mange QR-koder i bildet.

5.4 Anbefalt systemoppsett

Basert på Testoppsett 2, og testene som er utført i kap. 4, er det her satt opp et oppsett som er testet og fungerer. Dette er ikke ment som noen begrensing for systemet, men snarere et oppsett som raskt kan settes opp og vil gi resultater i det området som er omtalt i denne rapporten.

Tabell 8. Et systemoppsett som er testet i dette prosjektet

Kamera	PointGray BFLY-PGE-13E4C-CS
Redusert oppløsning	960x768 piksler
Objektiv	Brennvidde mellom 6 og 8 mm
Størrelse på QR-koder	20x20 mm
Avstand til QR-matrise	200 mm
Algoritme for å finne rotasjon	Snitt av alle linjer
Algoritme for å finne pikselstørrelse	*Ikke nødvendig*
Algoritme for å finne posisjon	Vinkelmetode
Kamerakalibrering og linsekorreksjon	*Valgfritt*
PC	Prosessor: Intel i5 3.1GHZ Minne: 16GB Ethernet: GB-hastighet
Forventet presisjon	Under ±0,3 mm. Marginalt forbedret med linsekorreksjon.
Forventet frekvens på posisjonsestimatene	Minimum 4Hz

5.5 Nedprioriterte deler og videre arbeid

I dette prosjektet har det vert hovedfokus på å finne algoritmer som fungerer godt til sine formål, deretter er de mest optimale algoritmene prøvd ut under ulike forhold. Det er selvsagt ting som kan påvirke estimatene og som ikke har blitt testet fullt ut. Her kommer en gjennomgang av noen av det som ikke er prioritert, og en liten grunngiving av hvorfor dette ikke er prioritert.

- **Ulike lyssettinger**
  Testene er gjort i et laboratorium med godt lys. Det er ikke gjort noen egen test for hvordan lyset påvirker estimatet. Dette ville i så fall påvirke antall QR-koder som blir detektert. Det finnes tester i kap. 4.4 og 4.9 som viser hvilken påvirkning antall QR-koder har på estimatene.

- **Støy i bilde**
  Det er ikke forsøkt å påvirke bildene med støy. Dette vil i tilfelle kunne påvirke antall detekterte QR-koder og nøyaktigheten til QR-kodenes bildekoordinater.

- **Ulike kamera**
  Det er ikke prøvd flere typer kamera enn det som er beskrevet i testoppsettene. Kameraene som er brukt er av anerkjente merker, og de er industrikameraer. Det kunne tenkes å se på hvilken presisjon billigere kamera ville gi. Det finnes veldig mange typer kameraer på markedet og i alle prisklasser. Skal det først testes, bør det gjøres på et større utvalg.

- **Teleobjektiv**
  Det er gjort tester som går på ulike brennvidder på objektivet. Det hadde vært svært ønskelig å se på om bruk av et teleobjektiv har påvirkning på resultatene. Det har innenfor prosjektperioden dessverre ikke vert mulig å få tak i teleobjektiv som passer til kameraet. I kap. 4.6 er det laget en test på brennvidder i vidvinkelområdet.

- **Flere avstander**
  Avstanden mellom kamera og motiv har vert 200 millimeter under testene. Dette er omtrent arbeidsavstanden til en sveiserobot, og det har blitt sett på som et utgangspunkt. Den direkte påvirkningen ved å øke eller minke avstanden er størrelsen en QR-kode utgjør i bildet. Den ideelle størrelsen til QR-kodene er testet ut i kap. 4.5 og vurdert både i millimeter, og som en prosentdel av bildet i kap. 5.2. Det vil da være enkelt å adaptere systemet til større eller mindre avstander.

- **Effektivisering av algoritmer for å øke frekvensen**
  Algoritmene er laget for å testes mot hverandre og for å finne presisjon. Behandlingstiden er sammenlignet for algoritmene, men de er ikke optimalisert for kortest mulig behandlingstid. Det er klart at et ferdig system som skal kunne kjøre online må optimaliseres, men det er ikke prioritert i dette prosjektet.

Videre arbeid

Denne oppgaven har funnet gode løsninger på det som har vert arbeidet med. Det er likevel områder som kan jobbes videre med.

- **Effektivisering av program**
  Alle algoritmene som er implementert i C++ i dette prosjektet er laget kun for å teste om, og hvor godt de fungerer. Det er ønskelig å optimalisere de algoritmene som har de best resultater til høyere prosesseringshastigheter.

- **Implementasjon i en applikasjon**
  Systemet som er laget, og det anbefalte systemoppsettet i kap. 5.4, er klart til å kunne implementeres i en fysisk applikasjon. Det kan dermed brukes som utgangspunkt til videre arbeid i ulike applikasjoner.

- **Effektivisere eller bytte ut ZBar**
  Biblioteket ZBar fra kap. 2.1, har fungert bra i prosjektet, men det er skanning av bildet med ZBar som bruker majoriteten av behandlingstiden i systemet. Det kan være mulig å effektivisere dette ved å se på lavnivået av biblioteket, og alternativt kan det sees på andre biblioteker. En mulighet som også kan jobbes med, er å lokalisere de områdene i bildet som inneholder QR-koder med andre bildebehandlingsmetoder, for deretter å segmentere bildet og skanne bare de områdene som inneholder QR-koder.

- **Andre alternativer for å fastsette pikselstørrelse**
  Pikselstørrelsen har vist seg å ikke være en så presis parameter som nødvendig, se kap. 5.1. Dette fører til at den algoritmen som ikke bruker pikselstørrelse fungerer langt bedre enn de andre i de fleste tilfeller. Det kan jobbes videre med å finne årsaken til dette og/eller andre algoritmer som gir et bedre estimat av pikselstørrelsen. Dette vil i så fall hjelpe Enkelmetode til å fungere enda bedre i de tilfellene den er foretrukket.

- **Andre kameraer**
  Det ville være interessant å se på hvor godt systemet fungerer ved å bytte ut industrikameraet med billigere webkameraer eller lignende. Testene av Vinkelmetode har vist god stabilitet gjennom hele kap. 4, og det gir indikasjoner om at kvaliteten på estimeringsgrunnlaget, med andre ord bildet, kanskje ikke er så viktig.

5.6 Konklusjon

Fra tidligere arbeid er det kjent at å estimere senterposisjonen til et bilde i romkoordinater er mulig ved å basere seg på punkt som er kjent i både i romkoordinater og i bildekoordinater. Denne rapporten viser at QR-koder kan brukes som referanse for punkteter i begge koordinatsystemer. Rapporten viser også at ved å velge optimale algoritmer for å estimere posisjon, er presisjonen til et posisjonsestimat på mindre enn ±2 ganger pikselstørrelsen, noe som har vert i størrelsesorden ±0,3 mm i dette prosjektets praktiske tester. Det er i rapporten foreslått et spesifikt testoppsett som fungerer med denne presisjonen. Det er med det lagt et vanntett grunnlag for å implementere systemet i de praktiske applikasjonene som er ønskelig.

Rapporten viser resultater basert på grundig testing, og det konkluderes med at den egenutviklede algoritmen Vinkelmetode er den generelt beste av de som er testet. Sammenlignet med de andre algoritmene, viser den spesielt gode resultater i de tilfellene hvor det er detektert få QR-koder i bildet. Denne algoritmen er også robust mot påvirkninger som linseforvrenging og reduksjon av oppløsning. Systemet er derfor lett anvendelig og kan settes opp enkelt uten å kalibrere kamera, eller å tenke nøye over oppløsning og andre parametere.

6. Referanser

[1] C. A. Linte, Z. Yaniv og P. Fallavollita, «Perspective- nPoint Problem,» i *Augmented Environments for Computer-Assisted Interventions*, Munich, Germany, Springer, 2015, pp. 52-.

[2] J. B. T. Alcoriza, «Camera location calculation using QR codes,» UiS, Stavanger, 2014.

[3] H. Sinnes og S. Hauge, «Automatisk kalibrering av verktøy for lakkbeholder,» UiS, Stavanger, 2017.

[4] DENSO WAVE INCORPORATED, «qrcode.com,» DENSO WAVE INCORPORATED, [Internett]. Available: http://www.qrcode.com/en/. [Funnet 30 01 2017].

[5] S. Freyer, *qrmat42.py,* Stavanger: UIS, 2012.

[6] J. Brown, «zbar.sourceforge,» GNU Lesser General Public License, 2008-2010. [Internett]. Available: http://zbar.sourceforge.net/. [Funnet 30 01 2017].

[7] Canon, «Capturing the image CCD and CMOS sensors,» Canon, [Internett]. Available: http://cpn.canon-europe.com/content/education/infobank/capturing_the_image/ccd_and_cmos_sensors.do. [Funnet 23 02 2017].

[8] S. Mchugh, «Understanding camera lenses,» Cambridge in Colour, [Internett]. Available: http://www.cambridgeincolour.com/tutorials/camera-lenses.htm. [Funnet 23 02 2017].

[9] Photography Mad, «STANDARD LENSES,» Photography Mad, [Internett]. Available: http://www.photographymad.com/pages/view/standard-lenses. [Funnet 23 02 2017].

[10] N. Mansurov, «photographylif,» photographylif, 11 6 2015. [Internett]. Available: https://photographylife.com/what-is-distortion. [Funnet 31 01 2017].

[11] Mathworks, «What Is Camera Calibration,» mathworks, [Internett]. Available: https://se.mathworks.com/help/vision/ug/camera-calibration.html. [Funnet 31 01 2017].

[12] OpenCV, «Camera Calibration and 3D Reconstruction,» OpenCV, 18 12 2015. [Internett]. Available: http://docs.opencv.org/3.1.0/d9/d0c/group__calib3d.html#ga687a1ab946686f0d85ae0363b5af1d7b. [Funnet 31 1 2017].

[13] openCV, «Camera Calibration,» openCV, 18 12 2015. [Internett]. Available: http://docs.opencv.org/3.1.0/dc/dbb/tutorial_py_calibration.html. [Funnet 31 1 2017].

[14] Z. Zhang, «A Flexible New Technique for Camera Calibration,» *IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 22,* p. 1330–1334, 1 1 2000.

[15] MathWorks Inc., «Computer Vision System Toolbox Functions,» MathWorks, Inc., [Internett]. Available: https://se.mathworks.com/help/vision/functionlist.html?s_cid=doc_ftr#bux207a. [Funnet 31 1 2017].

[16] openCV, «Geometric Image Transformations,» openCV, 15 12 2015. [Internett]. Available: http://docs.opencv.org/3.1.0/da/d54/group__imgproc__transform.html#ga69f2545a8b62a6b0fc2ee060dc30559d. [Funnet 31 1 2017].

[17] N. Berry, «datagenetics,» 8 2013. [Internett]. Available: http://datagenetics.com/blog/august32013/. [Funnet 1 2 2017].

[18] K. Matsuda og R. Lea, «Drawing and Transforming Triangles in WebGL,» i *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL*, Addison-Wesley Professional, 2013, pp. 67-113.

[19] GIS, «gisgeography,» gisgeography, [Internett]. Available: http://gisgeography.com/trilateration-triangulation-gps/. [Funnet 08 03 2017].

[20] W. Herman og W. S. Jr., «Determaination of a position in three dimensions using trilateration and aproximate distances,» Colorado School of Mines, Colorado, 1995.

[21] F. Reichenbach, A. Born, D. Timmermann og R. Bill, «Splitting the Linear Least Squares Problem for Precise Localization in Geosensor Networks,» University of Rostock, Rostock ,Germany, 2006.

[22] OpenCV, «operation on arrays,» opencv dev team, 07 03 2017. [Internett]. Available: http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#solve. [Funnet 08 03 2017].

[23] J. Traa, «Least-Squares Intersection of Lines,» UIUC, Illinois, 2013.

[24] openCV, «openCV,» OpenCV Developers Team: itseez, 2017. [Internett]. Available: http://opencv.org/about.html. [Funnet 1 2 2017].

[25] V. Lepetit, F. Moreno-Noguer og P. Fua, «EPnP: An Accurate O(n) Solution to the PnP Problem,» Springer Science + Business Media LLC, Lausanne, Switzerland, 2008.

[26] OpenCV, «OpenCV,» opencv, 22 5 2017. [Internett]. Available: http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp. [Funnet 23 5 2017].

[27] Trimble, «GPS toturial,» Trimble, [Internett]. Available: http://www.trimble.com/gps_tutorial/. [Funnet 07 Mars 2017].

[28] Lyndon B. Johnson Space Center, «Ultra-Wideband Angle-of-Arrival Tracking Systems,» NASA Tech Brief, Huston, 2010.

7. Vedlegg

Vedleggene ligger som digitale vedlegg i PDF-dokumentet.

MscQrPosV20

Siste versjon av kildekoden til algoritmer som har vært testet i prosjektet, skrevet i C++. I prosjektfolderen ligger det også ett testbilder som kan benyttes.

QR-matrise.pdf

En QR-matrise som er tilpasset å kunne skrive ut på et A3 ark og som kan brukes for test med kamera.

MscQrPos_test.mp4

En video som viser posisjonsestimat med kamera i bevegelse.
Grunnet stor filstørrelse er dette vedlegget lagt som en link til YouTube.
https://www.youtube.com/watch?v=32ao60naKok