2016

VRSS

Virtual Reality Surveillance System

BACHELOROPPGAVE MARINEINGENIØR ELEKTRONIKK OG DATA
ERLEND JAKOBSEN RØNNINGEN OG JOSTEIN KJERNAAS

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment

Formalia:

Linje:	Elektronikk & Data
År:	2016
Veileder:	Terje Fykse(Sksk) / Martin Vonheim Larsen (FFI) / Idar Dyrdal (FFI)
Antall sider:	69
Trykk:	Sjøkrigsskolen trykkeri

Oppgaveformulering

«Bacheloroppgaven skal gi offiserene anledning til å anvende kunnskaper og ferdigheter de har tilegnet seg ved bransjeutdanning ved Sjøkrigsskolen. Oppgaven skal gi erfaring i å arbeide med en problemorientert oppgave. Den skal gi øvelse i å gjennomføre et større arbeid alene eller i gruppe. Den skal gi offiseren tid til fordypning og trening i å løse teoretiske, eksperimentelle eller praktiske problemstillinger. Det skal legges vekt på å bruke tekniske problemstillinger fra Sjøforsvaret. Det kan være en oppgave gitt av Sjøkrigsskolen alene, i samarbeid med Forsvarets organisasjoner eller i samarbeid med sivil bedrift. Kadettene kan og fremme egne problemstillinger til avdeling for Teknologi for godkjenning.»
(Sjøkrigsskolen, 2013)

Forord

Denne besvarelsen tar for seg vår Bacheloroppgave ved linjen Marineingeniør Elektronikk og Data ved Sjøkrigsskolen. Med oppstart 20. Mars 2016 til levering 06. Desember 2016, strekker oppgaven seg over et tidsspenn på 8 måneder.

Oppgaven er utarbeidet av Jostein Kjernaas og Erlend Jakobsen Rønningen.
Jostein er tidligere alpinist og idrettsbefal ved KNM Harald Haarfagre, Madla. Han har videregående utdanning med spesialisering i realfag fra Ringerike videregående skole, samt grunnleggende befalsutdanning fra Madla, fullført i desember 2012.
Erlend er tidligere høydehopper og ansatt ved Oslo Børs VPS. Han har økonomiutdanning fra Høyskolen i Oslo og Akershus, samt grunnleggende befalsutdanning fra KNM Harald Haarfagre Madla, fullført i desember 2013.
Begge utdannes i dag til marineingeniører i elektronikk og data ved Sjøkrigsskolen, Bergen.

Denne oppgaven har for oss begge vært en særs spennende reise. Vi har for alvor fått et innblikk i teknologiens verden. Vi er i dag ikke i tvil om at denne teknologien er fremtiden. Vi håper at den lærdommen og erfaringen vi har tatt til oss er noe vi får bruk for. Ikke alt har vært like nyttig og relevant for vår tid som våpentekniske offiserer ombord på fartøy. Dog ser vi på all erfaring, uansett hvor relevant den så måtte være, som unik.

Vi har igjennom hele denne perioden fått særdeles god støtte fra vår veileder ved Forsvarets Forskningsinstitutt, forsker Martin Vonheim Larsen. Vi ønsker å uttrykke vår takknemmelighet og ydmykhet overfor Martin. Han har stilt opp til alle døgnets tider og vært upåklagelig hjelpsom og imøtekommende. For oss ville ikke denne oppgaven vært mulig å gjennomføre uten han.

Erlend Jakobsen Rønningen Jostein Kjernaas

Innhold

1 Sammendrag

Hensikten med denne bacheloroppgaven har vært å etablere en plattform hvor man kan få en situasjonsforståelse i en farlig situasjon, uten å eksponere seg selv. For oss har dette ført til en større forståelse og takknemlighet overfor de som tilrettelegger for at deres egne landsmenn skal være tryggere i jobben som soldat. Vår utførsel er en blanding av matematiske beregninger, programmering i C# og teori innen bildeprojeksjon.

I programmet vårt kan man, i sanntid, se et 360 grader liveoverført bilde av omgivelsene rundt seg med VR-briller. Man trer inn i en virtuell virkelighet, hvor plattformen er utformet av oss, basert på ønsker og krav fra operatørene som faktisk skal benytte brillene. Når operatøren observerer vil han føle det som om at han fysisk er i situasjonen uten at han eksponerer seg selv. Vi har lagt inn et brukergrensesnitt som skal gjøre det enklere for operatøren å ha oversikt i det virtuelle rommet. Et brukergrensesnitt utformet i samråd med operatører fra Kystjegerkommandoen og Marinejegerkommandoen.

Oppgaven beskriver en problemstilling vi har valgt å svare på ved hjelp av vår faglige kunnskap, teknologiske ferdigheter og nysgjerrighet. Alle diskusjonsforumene, forelesningsnotatene og artiklene vi har lest er i den hensikt å skape et best mulig produkt.

Målet vårt har hele tiden vært å lage en sammenkobling mellom VR-briller og 360 kamera med 1:1 bildeoverføring. Det som kameraet fanger opp, skal vises i VR-brillene samtidig. Videre skal hodebevegelsen føles naturlig. Vi er fornøyd med å ha nådd målet og vi ser et enormt utviklingspotensial for denne plattformen i fremtiden.

2 Introduksjon

Vi vil i introduksjonen greie ut om bakgrunnen for vår oppgave, herunder problemstillingen vi har valgt. Videre vil vi beskrive hvilke avgrensninger vi satte oss, samt hvilke ambisjonsnivå vi innledningsvis ønsket å legge oss på.

2.1 Bakgrunn

Da vi har det privilegium å bli utdannet sammen med norske operatører fra diverse spesialstyrker, kom vi i diskusjon med dem rundt fjernstyrte våpenplattformer. Det er systemer disse operatørene selv har erfaring med fra operativ oppdragsløsning. Blant alle fordelene de nevnte ved ubemannede eller fjernstyrte våpensystemer bet vi oss merke i en ulempe.

Helhetsoversikten til operatørene ble vesentlig dårligere hvis situasjonsbildet i stridssonen baserte seg på bilder fra en skjerm versus observasjoner gjort med det blotte øye. Denne observasjonen la grunnlaget for vår problemstilling i denne bacheloroppgaven.

2.2 Problemstilling

"Hvordan kan vi, som teknikere, ved bruk av teknologiske hjelpemidler, bidra til at våre medsoldater opprettholder helhetsoversikten, uten at de selv eksponeres."

Oppgaven ble i hovedsak gjennomført av Jostein og Erlend, med et team på skolen bestående av tidligere stridsbåt 90 sjef, sjef kystjegerkommandoen(KJK) og operatører fra både Marinejegerkommandoen(MJK) og KJK. Deres rolle var å bidra med praktisk erfaring slik at brukergrensesnittet stod i samsvar med det de mente var nødvendig ute i felten. Operatørene stilte også opp for testing, noe som var viktig med tanke på å avdekke problemer vedrørende sjøsyke og brukervennlighet i stressende situasjoner. På dette tidspunktet tok vi kontakt med Forsvarets Forskningsinstitutt (FFI) for veiledning og hjelp, se vedlegg 10.9 FFI søknad.

Vi kom i samsvar med teamet frem til en rekke nyttige sensorer og funksjoner som kunne programmeres inn i systemet for å gjøre jobben enklere for soldatene ute i strid.

Spesialstyrkene så ett behov for et slikt system, særlig i forbindelse med RWS, og vi mente potensialet var stort for at en slik plattform kunne videreutvikles til en virkelig suksess. Vi hadde en visjon om at VR-briller ville bli innført i Forsvaret i nær fremtid uavhengig av dagens situasjon.

Vi hadde hørt rykter om at våpenprodusenten Kongsberg hadde erfaring med at man kunne bli

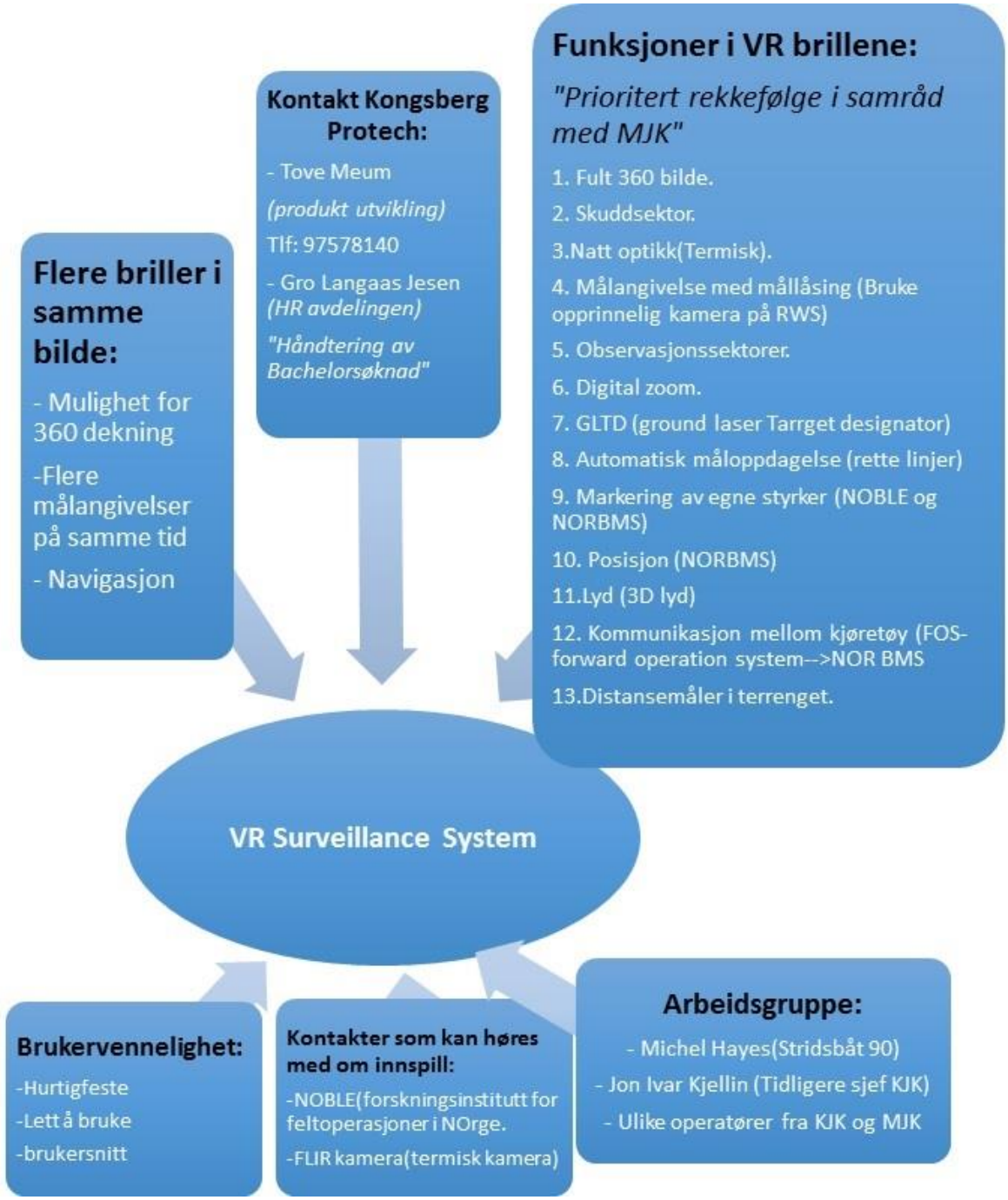
sjøsyk av å oppholde seg for lenge i en virtuell virkelighet. Likevel så vi for oss at dette problemet kunne løses ved å bruke et 360-kamera. I tillegg til at operatører fra MJK og KJK kunne trenes opp til å håndtere virtuell virkelighet over lengre tid.

2.3 Begrensing av oppgave

For å gjøre oppgaven oppnåelig innen tidsfristen ønsket vi å fokusere på kommunikasjonen mellom kamera og VR-brillene. Vi var klare over at i en operativ setting ville et slikt kamera trenge en gyrostabilisert plattform for å unngå vibrasjoner, men det hadde vi ikke kapasitet til å konstruere. Videre satte vi oss begrensinger innenfor brukegrensesnittet, og laget en smørbrødliste hvor vi huket av punkt for punkt avhengig av hvor mye vi rakk å gjøre på den tiden vi hadde til disposisjon.

2.4 Ambisjonsnivå

Vår idé og forslag til løsning på problemstillingen, var å konstruere et kamerasystem som opereres gjennom VR-briller. Tanken var å plassere et 360-kamera på våpenplattformen. Kameraet skulle sende live bilder av omgivelsene ned til en sikkert plassert operatør som fikk disse bildene opp i sine VR-briller. Bildet operatøren mottok ville dekke alle vinkler rundt våpenplattformen. Vi ønsket videre å programmere VR-brillene slik at operatøren kunne manøvrere seg rundt i bildet ved bruk av naturlige hodebevegelser. På denne måten kunne soldaten få følelsen av at han selv sitter oppe på våpenplattformen.



2.4.1 **Figur tankekart**.

Figuren viser en oversikt over tankekartet vi lagde helt i innledningsvis av oppgaven vår.

3 Teori

I vår oppgave benyttet vi 360 kamera og VR-briller for deretter å flette disse sammen med ulike programmer. Dette er ulike produkter som er utviklet basert på teknologisk utvikling og vitenskap. For å forstå helhetlig hvordan vårt prosjekt fungerer vil vi nå gå detaljert inn på vitenskapen bak disse produktene. Dette for å gi et lite innblikk i hva som faktisk skjer bak kulissene.

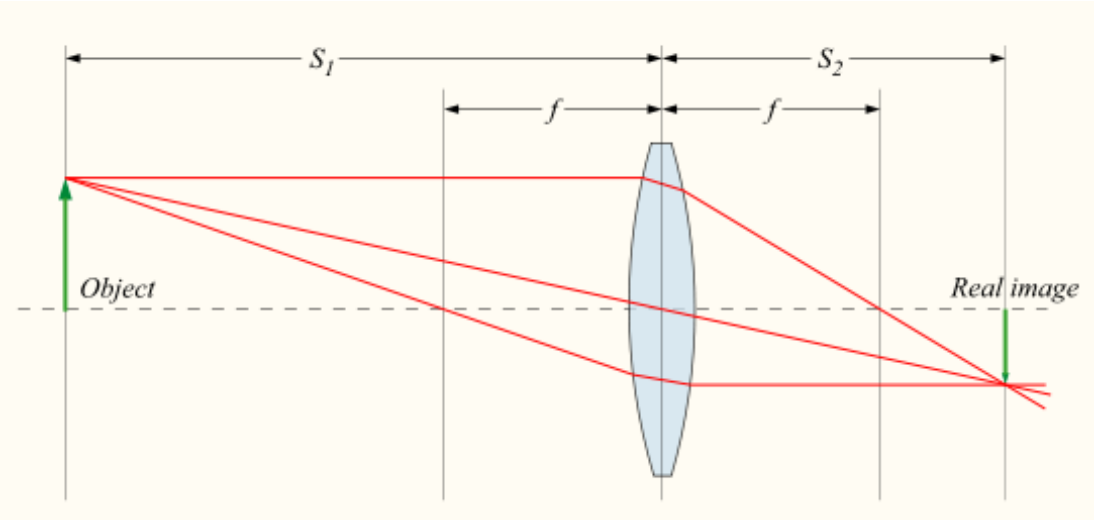
3.1 Fisheye

Kameraet vi brukte heter Theta Ricoh S, og har to fisheyelinser som sammen gir 360 graders synsfelt. En fisheyelinse er en ultra vidvinkellinse som produserer en sterk forvrengning av bilde. Dette gjør at den kan lage et vidt panoramabilde. Stort sett alle linser forvrenger virkeligheten slik at rette linjer ikke avbilde som rette. Fisheyelinser gjør derimot dette i veldig stor grad. Med en linsemodell prøver man å rette opp denne forvrengningen slik at rette linjer igjen avbildes som rette. Denne prosessen kalles for rektifisering. Forvrengningen er i tillegg størst langt ute til sidene, man vil derfor oppleve en dårligere oppløsning i denne delen av bildet. Fisheyelinser er problematisk å arbeide med, men når man har klart å utføre rektifiseringen har fisheyelinser ingen andre vesentlige ulemper.

3.1.1 Figur Fisheyelinse
hentet fra http://thefuturephotographer.com

Begrepet fisheye ble skapt i 1906 av den amerikanske fysikeren og oppfinneren Robert W. Wood, som baserte modellen på hvordan en fisk ville se en ultrawide halvkuleformet utsikt fra under vann (et fenomen kjent som Snell vindu). Fisheye er en linse som i stor grad minner

om en konveks linse siden den krummer utover på midten. Egenskapene til en konveks linse er at den bryter strålene som reflekteres av objektet innover mot sentrum(brennpunktet). I figuren under representeres brennpunktet som real image, hvor strålende samles etter brytning. Her vil objektet være speilvendt. Liten f utgjør brennvidden, som avgjør hvor strek brytningsevne systemet har. S1 og S2 representer avstanden fra henholdsvis objekt til linse, og linse til gjenskapt bilde.

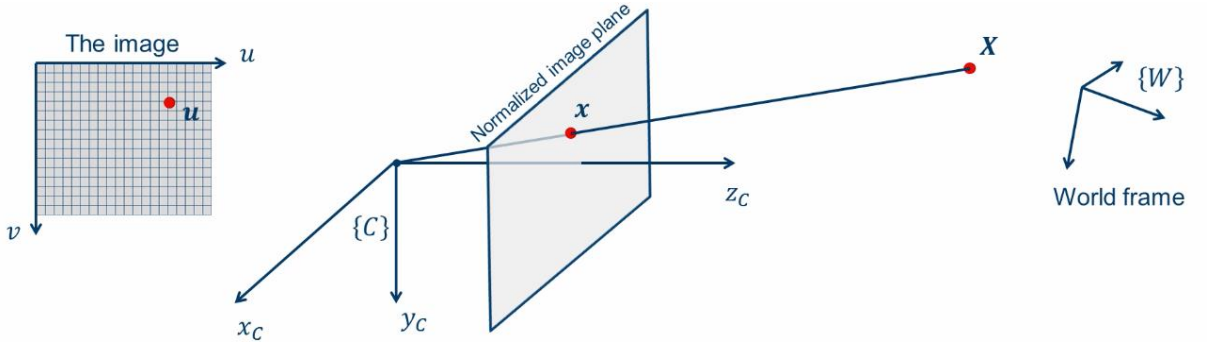


3.1.2 Figur Linsebrytning
Hentet fra https://.en.wikipedia.org

3.2 Perspektivkameramodellen

For å gå dypere inn i hvordan man fanger detaljene i virkeligheten til et bilde har vi analysert perspektivkameramodellen. Dette er en matematisk modell som beskriver sammenhengen mellom homogene punkter i virkeligheten og homogene punkter i bildet. Det er vanlig å beskrive perspektivkameramodellen i et kartesisk koordinatsystem. Kamerasenteret plasserer vi i origo, og lar z-aksen peke ut av kameraet, y-aksen peke nedover og x-aksen peke mot høyre. For å unngå speilvendte bilder innfører vi et virtuelt avbildningsplan (dvs. «sensoren» i kameraet) foran kameraet, som vist i figur 3.2.1 Perspektivkameramodell under. Det er vanlig å definere modellen i to steg. Først finner vi de *normaliserte bildekoordinatene* til X i punktet x der linjen mellom X og origo skjærer avbildningsplanet. Så finner vi pikselkoordinatene ved å justere for sensorstørrelsen.

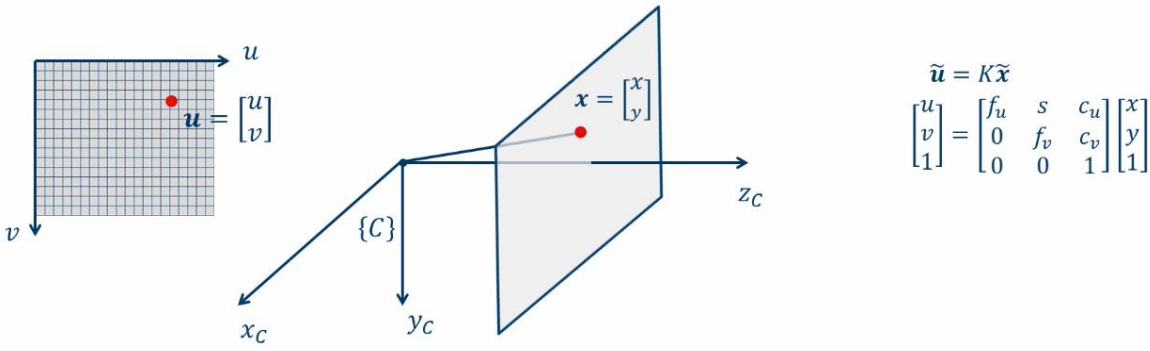
Ved å bruke homogene koordinater kan Perspektivkameramodellen representeres som to homogene matriser:



3.2.1 Figur Perspektivkameramodell
Hentet fra http://maskinsyn.unik.no

Matrise 1, representerer den intrinsikke delen av modellen. Denne går fra det normaliserte bildeplanet som figuren over viser, til selve bildet, altså pikselkoordinater her angitt ved u og v.

Her bruker vi K, en 3x3-matrise som også kalles kamerakalibreringsmatrisen. Det utføres en homogen transformasjon fra det normaliserte bildeplanet til selve bildet.



3.2.2 Figur Intrinsikke del
Hentet fra http://maskinsyn.unik.no

Størrelsene c_u og c_v står her for optisk senter, s tar høyde for skjevheten i kameraet. Størrelsen s settes i dagens moderne kamera lik null. Størrelsene f_u og f_v representerer avstanden mellom det projektive senteret til kamera og bildeplanet, altså skalerte varianter.

Matrise 2, [R t] utgjør den ekstrinsikke delen av modellen. Her gjøres egentlig to operasjoner i én. Først konverterer vi fra verdenskoordinatsystemet til kameraets koordinatsystem. Så gjør vi perspektivprojeksjonen inn i bildeplanet, som gir navnet til modellen. Dette er tydelig i figuren under, der vi ser hvordan matrisen er bygget opp av en euklidsk transformasjon og 3x4 projeksjonsmatrise.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{3\times3} & \boldsymbol{t}_{3\times1} \\ \mathbf{0}_{1\times3} & 1 \end{bmatrix}$$

3.2.3 Figur Matrisemultiplikasjon
Hentet fra http://maskinsyn.unik.no

Transformasjonen fra verdenskoordinater til kamerakoordinater er beskrevet av den blå matrisen. Dette er en euklidsk transformasjon som består av en 3x3-matrise R og en 3-vektor t. R representerer hvordan verden er rotert i forhold til kameraet, og t representer hvor origo i verden er i forhold til origo i kameraet.
Den røde 3x4 matrisen er en perspektiv projeksjon som tar oss fra 3D til 2D.
Ganger man 3x4 matrisen med en homogen 4-vektor ser vi at vi får følgende:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \\ 1 \end{bmatrix}$$

3.2.4 Figur Euklidsk vektor
Hentet fra http://maskinsyn.unik.no

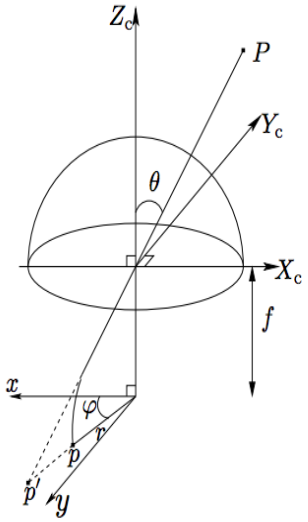
Dette er akkurat det vi trenger for å gjøre projeksjonen inn i planet z = 1. Satt sammen med T-matrisen ser vi at vi får [R t].

$$\tilde{\boldsymbol{u}} = K[R \quad \boldsymbol{t}]^W\tilde{\boldsymbol{X}} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} [R \quad \boldsymbol{t}]^W\tilde{\boldsymbol{X}} \quad \text{where} \quad \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} = {}^W\xi_C{}^{-1}$$

3.2.5 Figur Komplett Perspektiv Kamera modell
Hentet fra http://maskinsyn.unik.no

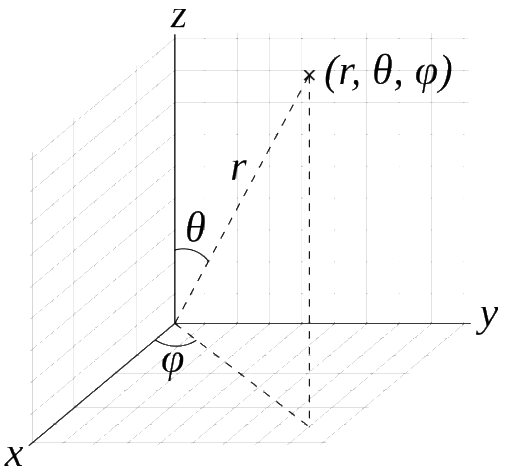
3.3 Fisheyelinsemodellen

Som vi nevnte i perspektivkameramodellen trenger vi en mer avansert modell for fisheye.



3.3.1 **Figur Fisheye**
Hentet fra https://rr.oulu.fi

Fisheyelinsemodellen er hentet fra artikkelen ***A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses,*** og sees på som stat of the art innen fisheyelinsemodeller. Programmet Skybox Shader implementerer denne modellen, ved å legge råmaterialet til kameraet på atmosfæren i det virtuelle rommet.
Her bruker vi sfære koordinater i stedet for kartesiske koordinater. I prinsippet starter man med å definere parameterne for kameramodellen. Videre transformerer man hvert punkt fra 3D-koordinater til bildekoordinater i piksler, for så å finne theta og pi for 3D punktet. Se figur:



3.3.2 Figur Theta – pi 3D
Hentet fra https://en.wikipedia.org

Deretter regnes radius ut basert på følgende ligning:

$$r(\theta) = k_1\theta + k_2\theta^3 + k_3\theta^5 + k_4\theta^7 + k_5\theta^9 + \dots \quad ,$$

hvor k representerer antall parameter.
Her brukes utregnet verdi av radius videre til å finne punktet i normaliserte bildekoordinater, uttrykt ved F(Φ)

$$\begin{pmatrix} x \\ y \end{pmatrix} = r(\theta) \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} = \mathcal{F}(\Phi),$$

3.3.3 Figur Normaliserte bildekoordinater
Hentet fra https://rr.oulu.fi

Theta Ricoh roterer bilde 90 grader, så i koden til Skybox Shader vil man se +pi/2 i utrykket.

3.4 Rendering

Rendering er en prosess som konverterer en datamodell til et bilde, som igjen kan vises på en skjerm. Disse datamodellene kan være 2D eller 3D, alt avhengig av hvilken beskrivelse modellen har. Med andre ord utfører rendering en dekorasjon av de objektene man har. Beskrivelsen til datamodellene kan være alt fra mur og sand til farge og refleksjonsgrad. For å lage bildet må gjenstandene plasseres i forhold til hverandre, og i forhold til den som skal se bildet. Programmet som utfører en rendering beregner farge og lysstyrke til hvert eneste punkt i bildet. Rendering kan også kalles resultatet av denne prosessen.

3.5 Hva er VR teknologi? Hva er virtuell virkelighet?

Kort historikk, ordet virtual reality dukket først opp i Frankrike i 1938, hvor Antonin Artuad beskrev teaterets virkemidler som "la réalité virituelle", som oversatt betyr virtuell virkelighet. Allerede i 1962 ble den første VR-maskinen konstruert, Sensorama. Maskinen viste korte filmer i 3D med vidvinkel og stereolyder. Dog slo ikke teknologien for alvor igjennom før i 2010, ledet av Oculus Rift. At Facebook kjøpte Oculus Rift for 2 milliarder dollar i 2014 sier sitt om at dette er noe som satses på.

VR teknologien funger slik at brillene skaper et virtuelt synsfelt hele veien rundt bæreren. Det er to skjermer inne i selve brillen, altså er brillen stereoskopisk, en skjerm foran hvert øye. Disse to skjermene skaper dybde og 3D-opplevelse i bildet. I brillen er det et akselerometer som registrerer hodets bevegelser og sørger for at du kan se rundt i det virtuelle rommet. Ved at bilde er noe ulikt i hver av skjermene lurer man hjernen til å tro at det bilde man ser er det samme som det dybdesynet og det samlede sanseinntrykket skaper av den virkelige verden.

For å ha en god opplevelse er det viktig at bilde i skjermen oppdaterer seg hurtig. At man har et visst antall bilder per sekund. De to største aktørene i markedet Oculus Rift og HTC Vive har begge en framerate, eller refresh rate om du vil på 90Hz. Dette vil si 90 bilder per sekund. Hz er en måleenhet for frekvens, derav antall svingninger/hendelser per sekund. Hvor god oppløsning som sitter i brillene er videre en viktig faktor for god brukeropplevelse. Både Rift og Vive har en oppløsning på 2160x1200 piksler. I selve brillene sitter et akselerometer gyroskop, Lighthouse laser tracking system, front-facing camera. Alle data overføres via HDMI USB 2.0 og 3.0. Å kombinere akselerometer, gyroskop og lighthouse laser tracking system er en måte som gjør at man får god tracking. Et annet eksempel er å ha et kamera på brillene som detekterer kjente markører ute i rommet som man sporer. Det viktigste er uansett at man har en nøyaktig metode som finner ut hvor brillene er og i hvilken retning de peker.

3.6 Akselerometer

Et akselerometer er et utstyr, indusert av tyngdekraften som registrerer eller reagerer på endringer i fart og retning. Det er bygd opp slik at det har en frittbærende stang med en masse og elektronikk som måler forskyving.

Brukes ofte sammen med et gyroskop, se neste avsnitt, for å navigere og styre blant annet fly, raketter, ubåter og droner. Baserer seg på Ekvivalensprinsippet utarbeidet av Albert Einstein, som sier at alle punkter som er i rommet har lokale utlikninger av et gravitasjonsfelt som er identiske med de lokale utlikningene av en uniform akselerasjon. I praksis betyr dette at massen inne i akselerometeret som beveger seg fritt vil bli forskjøvet så langt som akselerometeret beveger seg. Videre vil den frittbærende stangen være fast mot den flaten man måler opp mot. Utslaget i masse som beveger seg fritt måler akselerasjonen. Akselerometer kan måle langs alle tre aksene.

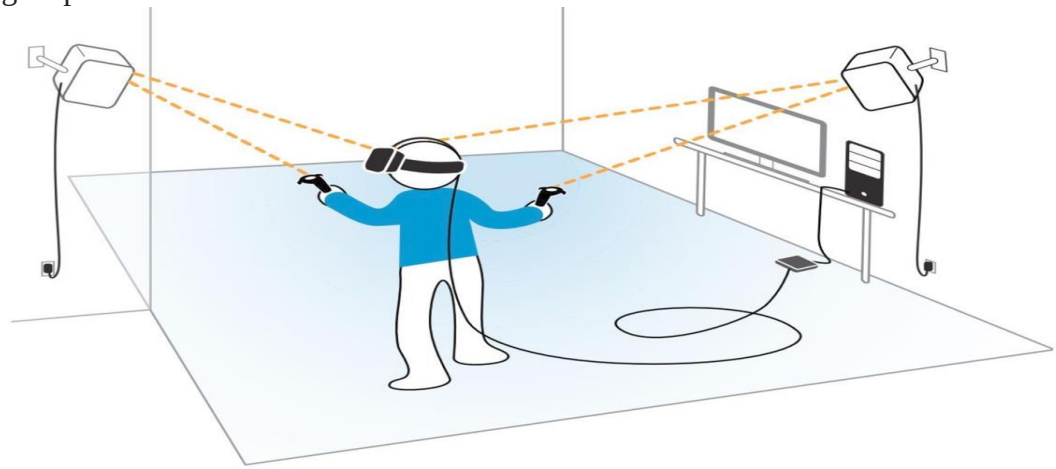
3.7 Gyroskop

Gyroskop er et svinghjul festet på en aksling som igjen er festet med to dreielagre i en ytre ramme. Svinghjulet har et stort treghetsmoment og roterer om symmetriaksen. Svinghjulet kan rotere om den symmetriske aksen med stor hastighet. Settes dette svinghjulet i hurtig rotasjon beholder rotoraksen sin retning i rommet, som igjen er et eksempel på prinsippet om bevaring av bevegelsesmengde. Brukes i instrumenter for retningsstabilisering, indikering og kontroll.

3.8 Light house tracking technology

Light house tracking technology er produsert av den amerikanske spillprodusenten Valve. Lighthouse tracking er måten brillene navigerer seg i rommet på. Det baserer seg på samme prinsipp som man finner i et lysthus. De to stasjonene som benyttes har ingen tilkobling til datamaskinen eller programmet, alt registerets i brillene. De to stasjonene er utstyrt med LED og laseremittere som sender ut 60 blitz pr sekund. LED sender først ut ikke-synlig lys, deretter sender laseremitteren ut en laserstråle som dekker hele rommet. Brillene bruker deretter disse fotosensorene til å detektere lys og laserstrålen. Når strålen treffer brillene begynner en tellemekanisme inne i brillene å gå, helt til den ser hvilke av fotosensorene på brillene som har blitt truffet av strålen. Den bruker deretter forholdet mellom hvor og når lyset traff brillene til å kalkulere den eksakte posisjonen til brillen i rommet. Ved at nok fotosensorer på brillene blir truffet av laserstrålen kan brillene skape en 3D simulering av

virkeligheten. Ikke bare til å kartlegge hvor brillene er på et gitt tidspunkt, men også i hvilken retning de peker.



3.8.1 Figur Light house tracking
Hentet fra www.tek.no

3.9 Kalmanfilter

I og med at disse LED og laseremitterne gir målinger i 60Hz må skjermene i brillene med oppdateringsrate på 90 Hz interpolere. Her utfører brillene en prediksjon av hvor den neste posisjonen til brillene kommer til å være litt frem i tid. Dette skjer hver gang man skal tegne til brillene, basert på et Kalmanfilter.

Kort forklart er et Kalmanfilter en algoritme som bruker målinger over tid til å produsere et estimat for en mer nøyaktig måling. I vårt tilfelle er det et estimat for neste posisjon. Tanken er at ved å ha et estimat for prosessens tilstand i tidspunkt t_{k-1}, vil man kunne gi et foreløpig (a priori) estimat av prosessens tilstand ved tidspunkt t_k, før denne målingen er tilgjengelig. Hovedlikningen til Kalman:

$$\hat{y}_k = w_k \cdot z_k + (1 - w_k) \cdot \hat{y}_k \, .$$

3.9.1 Figur Kalmanlikning
Hentet fra Notat Knut Meen.

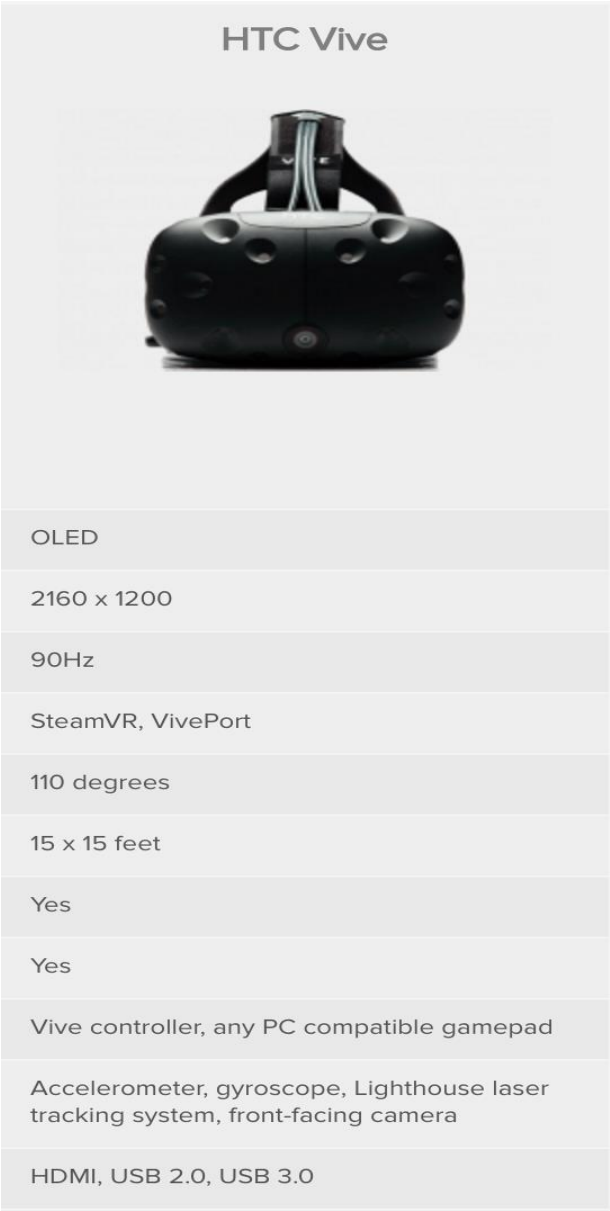
hvor $y_k\hat{}$ er aposteriori estimat etter måling, og $y_k\hat{}$. er apriori estimat før måling. Liten z_k er en observasjon som inneholder en stokastisk feil v_k. Vekten w_k blir bestemt slik at prediksjonsfeilen til $y_k\hat{}$ er så liten som mulig.

4 Ressurser

I denne oppgaven har vi tatt i bruk utstyr som allerede eksisterer og som er ute på det åpne markedet. Vi har basert oss på å sette sammen ulike produkter andre har skapt, slik at summen blir noe nyttig og unikt. Under ressurser vil vi greie ut om de fysiske innkjøpene (hardware) vi har gjort, samt hvilken programvare (software) vi har benyttet.

4.1 HTC Vive

Da vi skulle velge briller å jobbe med i denne oppgave stod valget i all hovedsak i mellom HTC Vive og Oculus Rift. Sistnevnte har vært banebrytende og ikke minst ledende innenfor VR-teknologien. Dog har de fått mer og mer konkurranse de siste årene. Oculus Rift og HTC Vive har begge samme oppløsning 2160x1200(1200x1080 pr øye) i de to skjermene som sitter i selve brillene. Begge har OLED display og viser bilder med en frekvens på 90Hz. Begge har et synsfelt på 110 grader. For å ta den endelig avgjørelsen måtte vi derfor spørre oss selv, hva gir best opplevelse? Hvilken har best tracking? Hva skiller disse to, siden oppdateringsraten og oppløsningen er den samme? Vi kom etterhvert frem til at HTC Vive ville gi en bedre opplevelse siden den har bedre tracking. Dette baserte vi på de anmeldelsene som lå ute på gitt tidspunkt, hvor blant annet light house laser tracking system ble fremhevet. Det at HTC Vive også anvender Kalman filter som vi lærer om i Digital signalbehandling ved Sjøkrigsskolen 5. semester spilte en faktor. Vi så her en mulighet til å knytte vårt prosjekt opp mot faglig innhold.



4.1.1 Figur HTC VIVE

Hentet fra www.digitaltrends.com

HTC Vive krevde en del av datamaskinen, både i forhold til CPU, operativsystem, grafikkort og minne. HTC oppgir i dag at man bør ha en prosessor med Intel

i5-4590, AMD FX 8350 eller bedre, samt minimum 4 GB i RAM i minne. Videre kreves det et grafikkkort som minimum er lik Nvidia GeForce GTX 1060, AMD Radeon RX 480, helst bedre. Som operativsystem må man kjøre Windows™ 7 SP1, Windows™ 8.1 eller Windows™ 10. Alle disse kravene gjorde at vi måtte kjøpe en datamaskin vi faktisk kunne jobbe med. Vi kjøpte derfor en JORAH I626S FATA1TY KILLER, med GeForce GTX 1070 8GB skjermkort, en 6. gen. Intel Core i7-6700 prosessor og 8GB DDR4 RAM 2133MHz i minne.

4.2 Ricoh Theta S

Da vi skulle velge kamera var det viktig for oss at kameraet var så enkelt som mulig. Dette grunnet tenkt scenario, man får en treffer ute på patrulje slik at kameraet blir satt ut av spill. For en operatør skal man bare kunne dra frem et nytt kamera, koble til kabelen og sette det på taket og man er oppe og går igjen. Andre viktige spesifikasjoner var at kamerat skulle kunne filme 360 grader med en livestream funksjon. I tillegg til at det måtte være mulig å hente ut rådata til videofila. Samsung Gear 360 var et alternativ vi faktisk kjøpte inn, dog var det ikke mulig å hente ut rådata. Alt av livestream ble sendt via wifi og måtte kjøres igjennom et program utviklet av Android. Dette betyr at Windows ikke registrerte kameraet som et webkamera, noe Windows kun gjør hvis det har tilgang til rådata. Alternativt kunne vi hacket Androidprogrammet manuelt og hentet ut rådataen. Dette så vi derimot ikke på som en løsning da det ville tatt mye tid og kapasitet.

Vi endte til slutt opp med Ricoh Theta S. Kameraet lot oss umiddelbart hente opp rådata, uten noe mer

Ricoh Theta S

Størrelse: 4,4 x 13 x 2,29 cm
Vekt: 125 gram
Skjerm: Nei, indikatorlamper
Innebygget lagring: 8 GB, ikke plass for minnekort
Videostøtte: Full HD, 30 fps
Kamera: 2 x 12 Mp, leverer 14 Mp ferdig stillbilde
Opptaksmoduser: Sfærisk stillbilde, video, manuell kontroll over lukkertid og mer
Batteri: Holder til 260 bilder per lading, eller 25 minutter opptak.
Tilkobling: WiFi, Micro USB, Micro HDMI
Stativfeste: Ja
Fuktsikret: Nei

Støtter: Android og iOS
Pris: 4000 kroner

4.2.1 Figur Ricoh Theta S

Hentet fra www.lydogbilde.no

innstallering av programvare på maskinen. At kameraet lot oss strømme direkte 360 video, gjorde det enkelt å hente opp rådata i C# med en webcam funksjon. Som figur 4.4.1 viser består kameraet av to fisheye objektiver plassert på hver sin side av kamerahuset. Disse objektivene har en blenderåpning på F/2,0 med 1/2.3" 12 Megapixel CMOS-sensor. Kameraet har en innebygd lagringsplass på 8 gigabyte.
Vi så i tillegg en mulighet med dette kameraet til å tilføre oppgaven enda mer tverrfaglig tyngde.

I faget elektromagnetisme ved Sjøkrigsskolen 4. semester lærte vi om brytning av stråler med konvekse linser. Dette er teori som kan anvendes på svært mange 360 kamera i dag. To objekter gjorde derimot at vi måtte stifte 360 bildet sammen. Den matriseregningen som utfører dette i renderingprogrammet har vi hatt en dypere forståelse for takket være undervisningen matematikk 3.

4.3 Unity

For å kunne redigere en virtuell verden trengte vi et program hvor man kan skape en plattform. Vi hadde flere alternativer og velge imellom, men endte til slutt ned på to kandidater, Unity 5(game engine) og Unreal Engine. Begge er spillutviklingsprogrammer som opprinnelig er tenkt for å utvikle dataspill på ulike spillplattformer. For oss som har lite erfaring med å lage visuelle spillplattformer var det viktig at programmet var så intuitivt som mulig. Vi valgte Unity 5 for at det i stor grad har et veldig visuelt preg. Man kan eksempelvis dra å redigere kuler og plan, uten at det er nødvendig og måtte skrive kode. Vi så her en mulighet til å hurtigere utvikle et brukergrensesnitt, og at det derfor ville være mer passende for vår oppgave.

4.3.1 Figur UNITY
Hentet fra htttps://en.wikipedia.org

4.4 C Sharp

For å programmere i Unity 5 kan man benytte ulike programmeringsspråk. For oss var det naturlig å velge C++, da vi har hatt undervisning i dette programmeringsspråket ved Sjøkrigsskolen over to semester. I så tilfelle måtte vi ha brukt openVR, hvor alt av grafikk, rendering og programmering av brillene måtte blitt gjort fra bunn av. Dette ble vi derimot frarådet av vår veileder ved FFI. Han mente at med våre ambisjoner om brukergrensesnitt kunne vi med fordel programmere i C#, som er kodespråket Unity er konstruert på. C Sharp, som det uttales, er et objektorientert programmeringsspråk basert på C++ og Java, utviklet av Microsoft. Når man programmerer med C# i Unity bruker man Visual Studio. Fordelen med C#, spesielt i vårt tilfelle var at man har muligheten til å bruke biblioteker og datatyper skrevet av andre til å utvikle sitt eget prosjekt. Ved å velge C# sparte vi oss selv utallige timer frustrasjon og flere tusen linjer med koding. Vi ville ikke kommet i nærheten av der vi er i dag hvis vi skulle programmert i C++.

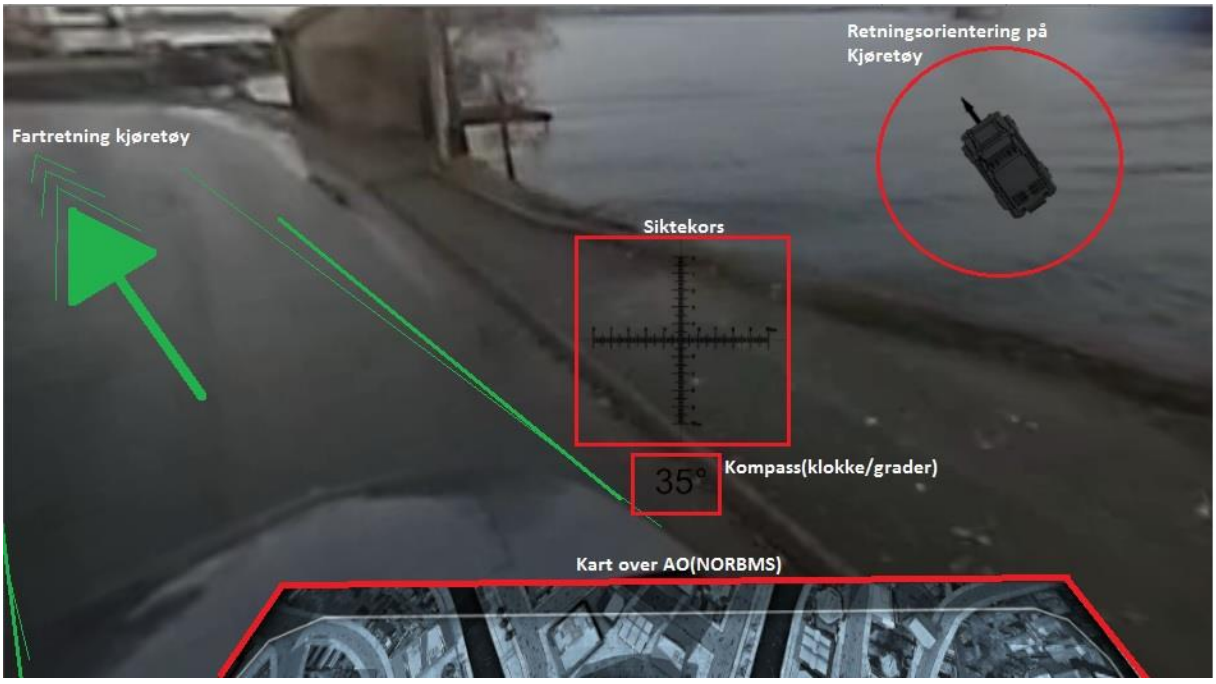
5 Utvikling

I denne delen av oppgavene skal selve utviklingen av produktet presenteres. Helt fra begynnelsen av prosjektet har vi satt oss delmål. Totalt bestod prosjektet vårt av 11 delmål. På denne måten følte vi at vi fikk strukturert arbeidet vårt på en god måte.

I utviklingen vil vi begynne med en forholdsvis kort presentasjon av hvordan det endelige produktet er satt sammen. Videre følger en mer detaljert punktvis gjennomgang av de 11 delmålene i kronologisk rekkefølge. Rapporten går ikke helt i dybden på hvert enkelt punkt, men prøver å gi leser en god forståelse av fremgangsmåten i de ulike delmålene. Programmeringsskriptene ligger ved som vedlegg, og derfor er minimalt av koding i C# tatt med i utviklingen. Underveis vil utviklingen også henvise til teori der dette er hensiktsmessig.

5.1 Ferdig Produkt

Figur 5.1.1 viser et øyeblikksbilde tatt fra TEST 3, slik operatøren ser det virtuelle rommet i brillene.



5.1.1 Figur Ferdig produkt

Vårt innlagte brukergrensesnitt er innrammet i rødt, mens kjøretøyets fartsretning er merket med grønt. Her har operatøren oversikt over:

- Senter i brillene (siktekors).
- Retning på kjøretøy i forhold til synsfelt.
- Retning på siktekors i forhold til kompass.
- Kart (Battlefield Management System).

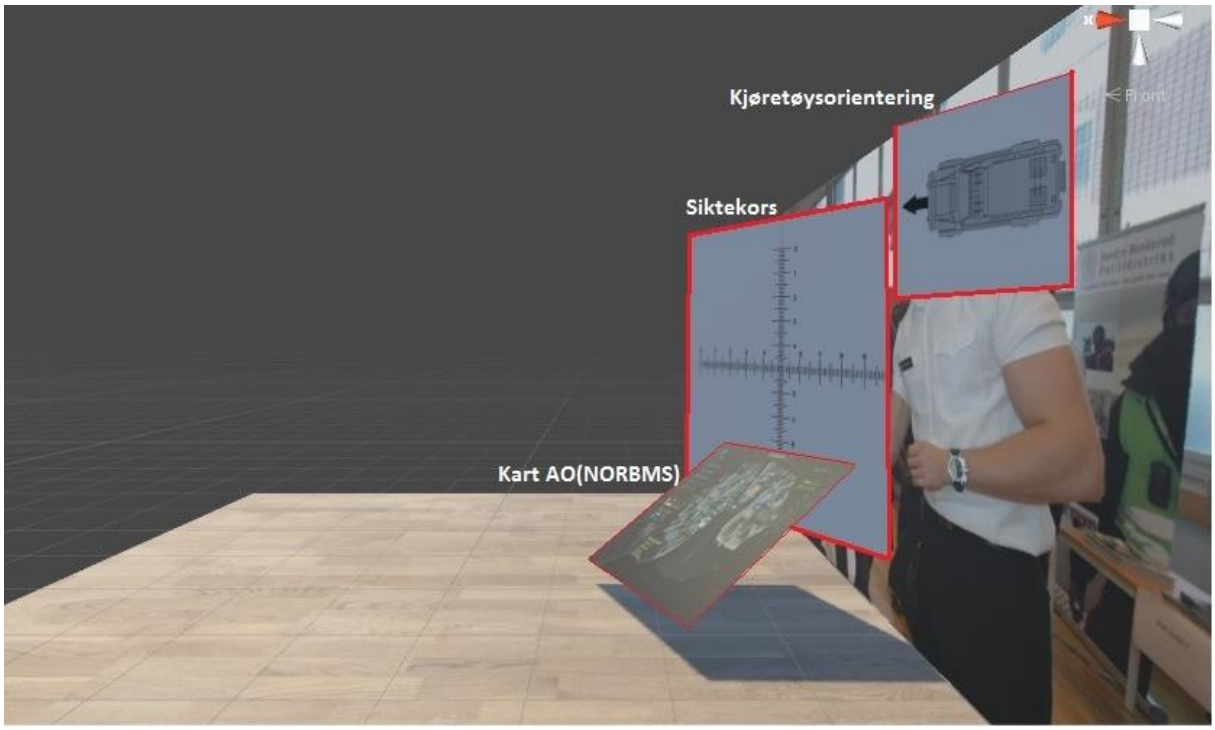
Dette er alle plan som er lagt til i det virtuelle bildet for å gi operatøren nyttig informasjon.

Kompassfunksjonen kan ved et tastetrykk endres til klokkeanvisning i forhold til kjøretøyets fartsretning. Dette ble lagt inn som ønske fra operatører i MJK, da de foretrekker å bruke klokkeangivelse fremfor grader når de orienterer seg i situasjoner. Grader brukes derimot når operatøren skal styre ildstøtte fra luften. Det er en viktig forskjell i de to funksjonene på det teknologiske planet;

- Grader tar utgangspunkt i himmelretningene (kjøretøyets kompass).
- Klokkemetoden ser på forholdet mellom synsfelt og kjøretøyets fartsretning.

Kart over operasjonsområdet orienteres alltid etter synsretningen i brillene. Dette gjør at det er svært intuitivt å kjenne seg igjen i kartet. Du vet alltid hvordan du er orientert, fordi retningen på kartet forblir den samme selv om operatøren vender blikket i andre retninger. Programmet har også en innlagt gjennomsiktighetsfunksjon slik at lysstyrken på karten kan stilles manuelt. Dette kan benyttes for å endre kartets lysstyrke etter omgivelsene ved henholdsvis natt og dag. Funksjonen kan også stilles slik at kartet forsvinner helt om ønskelig. Dette gjøres ved å stille lysstyrken helt til null. Størrelsen på selve kartplanet blir skalert etter kjøretøyets størrelse. Når operatøren er ute på oppdrag trenger han ikke å se takkonstruksjonen på eget kjøretøy, derfor brukes dette området til kartplan. I produktet har vi festet på et virtuelt Battle Map System, men når produktet skal tas i bruk ønsker Marinejegerkommandoen at Norwegian Battlefield Management System(NORBMS) skal vises på dette planet. NORBMS er et kommunikasjonssystem som norske styrker bruker i militære operasjoner. Med dette systemet vil operatøren til enhver tid ha oversikt over både sin egen posisjon, allierte og eventuelle fiendtlige posisjoner.

Figur 5.1.2 viser oppbygning av brukergrensesnittet i brillene.



5.1.2 Figur Planoppbygging

Hvert av disse hjelpemidlene ble konstruert hver for seg i sine respektive plan. Disse planene ble deretter lagt på råmaterialet som kom inn fra 360 kameraet i ønskede posisjoner. Deretter ble de ulike planene rotert i forhold til de tilknyttede referansepunkt.

5.2 Delmål 1 Forstå oppkobling av HTC Vive

Det var ikke bare å ta ut HTC Vive brillene fra boksen, plugge i en maskin og begynne å bruke produktet. Før produktet kunne tas i bruk var det en del programvare som måtte lastes ned. For å benytte seg av VR-brillene i spillsammenheng måtte vi laste ned følgende:

- **Steam VR**
  https://support.steampowered.com/kb_article.php?ref=5254-FJKZ-7829
- **Steam** (opprett bruker i Steam)
  http://store.steampowered.com/about/
- **Vive**
  https://www.htcvive.com/us/setup/

Disse programmene gjorde at man kunne laste ned et demospill fra Steam og ta i bruk brillene for første gang. For vår del skulle vi ikke bruke brillene i spillsammenheng og måtte på dette tidspunktet bestemme oss for hvilket utviklingsprogram vi ønsket å jobbe i. Vi valgte å programmere i Unity 5.4, med Visual Studio og C# som programmeringsspråk. Henviser til 4.3 under programvare.

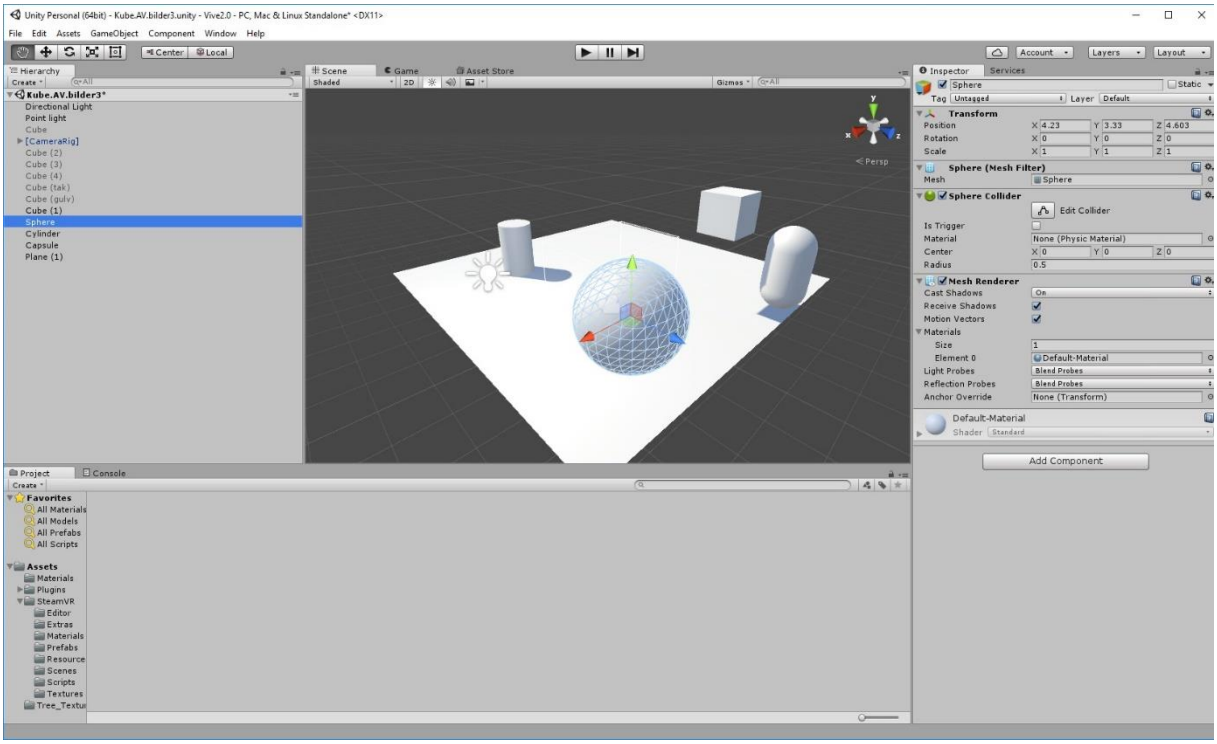
Vi lastet ned følgende programmer:

- **Unity 5.4**
  https://store.unity.com/?_ga=1.240372042.814995130.1469539933
- **Visual Studio.**
  https://www.visualstudio.com/downloads/

5.3 Delmål 2 Hvordan konstruerer noe eget i Unity

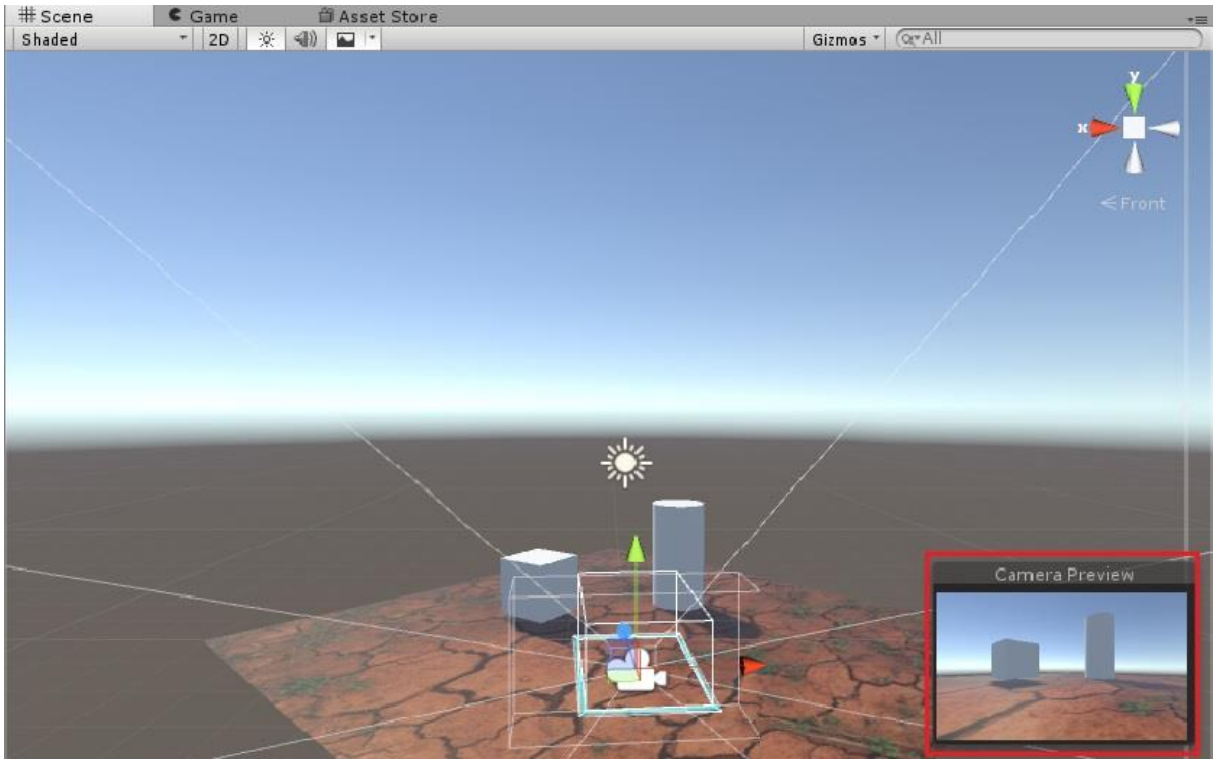
For å benytte seg av VR-brillene i tilknytning til Unity, lastet vi ned "STEAMVR_plugin".
https://www.assetstore.unity3d.com/en/#!/content/32647

Dette er et bibliotek som gjør det mulig å vise det virtuelle rommet som konstrueres i Unity i VR-brillene. For å få et innblikk i hvordan dette virtuelle rommet fungerte lærte vi oss å konstruere ulike geografiske figurer.



5.3.1 Figur Geometriske figurer

I funksjonen «Gameobject» gir Unity deg muligheten til å konstruere dine egne 3D objekter. Figur 5.1.3 viser et bilde av 5 ulike geografiske 3D figurer som vi konstruerte. Ved hjelp av disse figurene fikk vi kunnskap om hvordan man roterte, endret posisjon og størrelse på figurene.



5.3.2 Figur Camera Preview

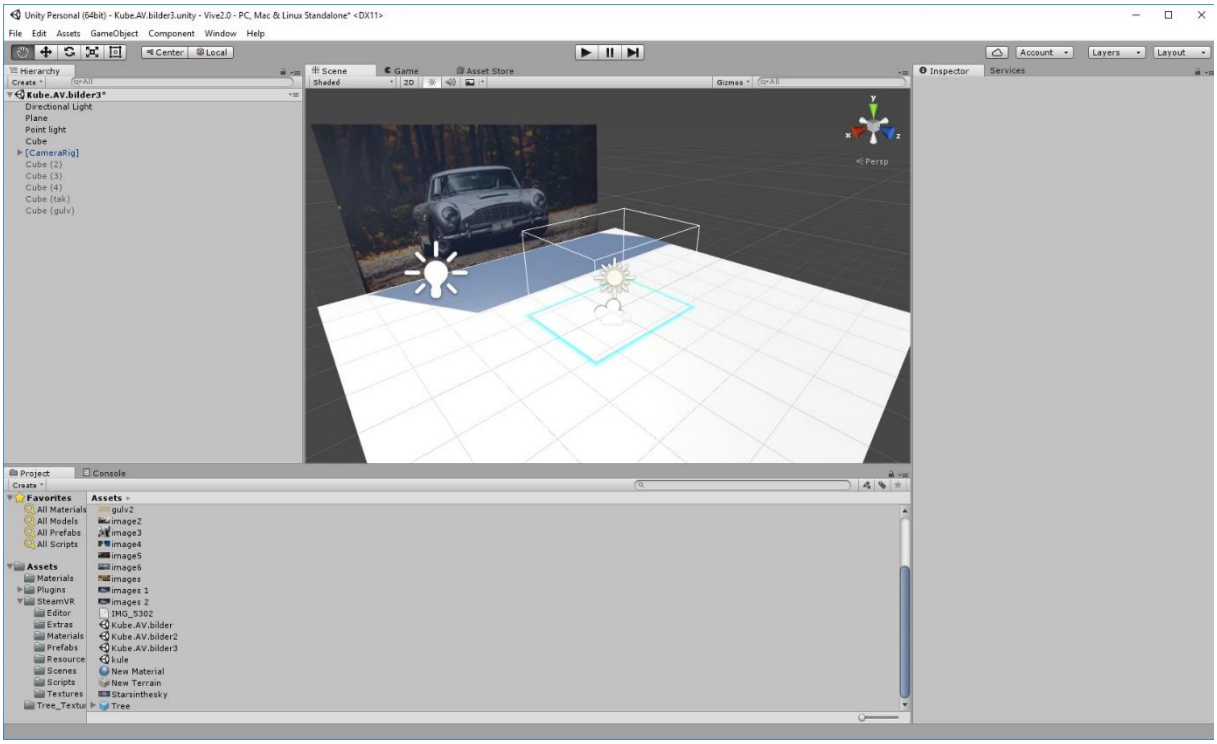
Figur 5.3.2 viser hvordan vi kan styre posisjonen til VR-brillene i det virtuelle rommet. I vinduet "Camera Preview" ser vi hvordan bildet vises i VR-brillene. Selve kameraposisjonen (merket med blå firkant) kan flyttes rundt i det virtuelle rommet, slik at de samme 3D objektene kan oppleves fra forskjellige perspektiv.

5.4 Delmål 3 Konstruere et lerret i Unity

For å forstå hvordan et 360 graders bilde presenteres i VR-brillene valgte vi å bygge opp forståelsen lag for lag. Tilslutt hadde vi en fullstendig forståelse av prosessen og på den måten fikk vi en mye større forståelse for potensialet til produktet videre. Vi bestemte oss for, i samarbeid med veilederen vår, å begynne med å konstruere et lerret. Deretter å bygge på med mer komplekse design etterhvert.

For å konstruere et lerret i Unity, benyttet vi oss av to plan. Et av planene roterte vi 90 grader om x-aksen og plasserte på kanten av det andre planet (veggplan). På denne måten simulerte vi et plan som ble gulv, og et plan som tilsvarte en vegg. For at konstruksjonen vår skulle tilsvare et lerret, var vi nødt til å hente inn et bilde som vi kunne dekorere veggplanet med.

Figur 5.4.1 viser en oversikt over lerretet.



5.4.1 Figur Lerret

Disse veggdekorene lagde vi på to forskjellige måter.

- Metode 1:
  Unity har en egen funksjon som beklèr et plan automatisk. Bildet du ønsker å beklè planet med legges inn under mappen "Assets" i Unity, deretter drar man bare bildet inn på det ønskelige planet.

For å forstå hvordan Unity gjorde denne prosessen valgte vi å utvikle et skript som utførte akkurat samme oppgave for oss.

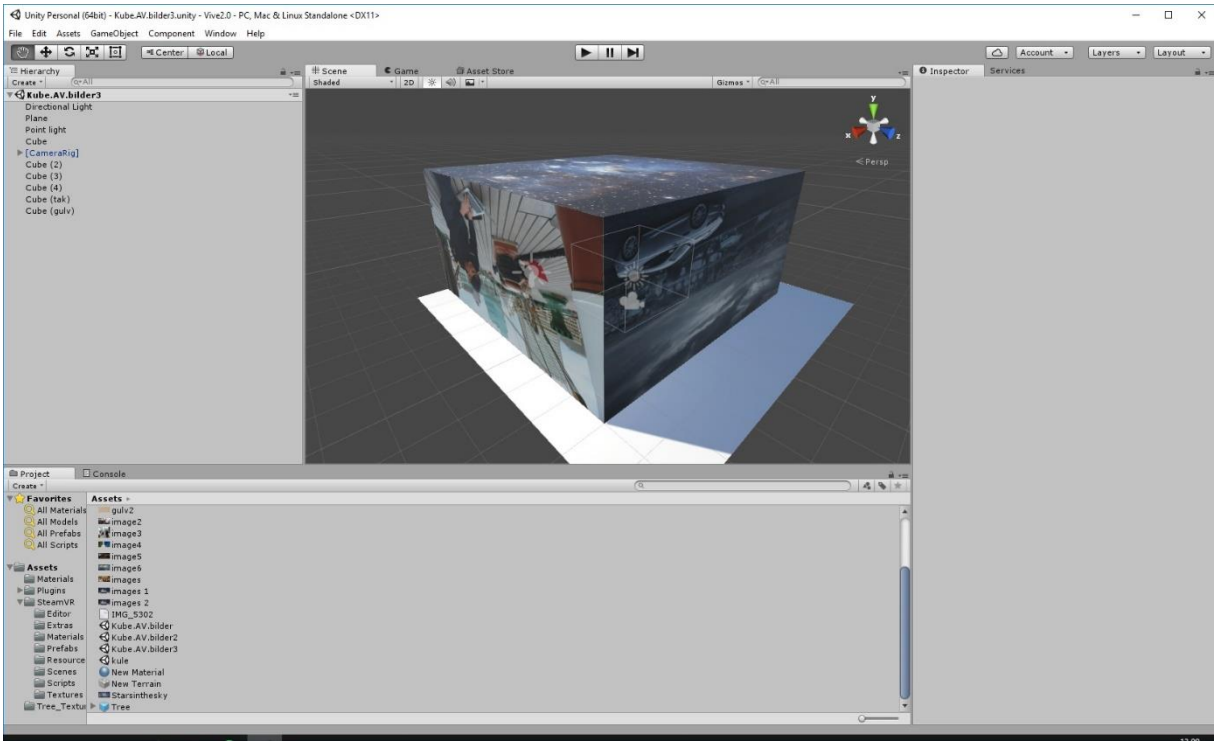
- Metode 2:
  I C# konstruerte vi et program som hentet inn en fil fra Datamaskinens harddisk. Med dette bildet fikk vi skriptet til å dekorere ønskelig plan for oss. Et bilde i Unity blir sett på som en «texture». Ved å bruke bibliotekfunksjonen i C# som kalles «Renderer» kunne vi dekorere planet.
  Se Vedlegg 10.1 Manuell bildeinnhenting.

Problemet med denne løsningen opplevde vi når prosjektet vårt ble jobbet med fra to forskjellige maskiner. Denne løsningen krevde nemlig at alle bilder som benyttes i prosjektet ligger på harddisken på akkurat den maskinen prosjektet kjører på. Vi så på det som unødvendig mye arbeid å legge inn alle bildefiler vi brukte på harddisken til samtlige av maskinene vi arbeidet fra.

I metode 1 derimot, kunne vi legge alle filer inn under Unity biblioteket og dermed fulgte filene automatisk med da prosjektet ble flyttet fra en maskin til en annen.
I tillegg så vi på bruk av allerede utviklet teknologi som svært relevant når vårt produkt skulle utvikles. Vi så ingen poeng i å finne opp nye metoder å utføre en handling som det allerede var funnet opp metoder for å løse. Metode 2 ga oss en innsikt i hvordan prosessen i Unity utføres, dette tok vi med oss som nyttig lærdom videre i prosjektet, men vi bestemte oss for å bruke metode 1 hvis liknende situasjoner skulle dukke opp igjen.

5.5 Delmål 4 Konstruere ett bildegalleri i Unity

Når et lerret var på plass, ønsket vi å gjøre det litt mer komplekst ved å lage et rom dekorert med bilder på tak, vegger og gulv.

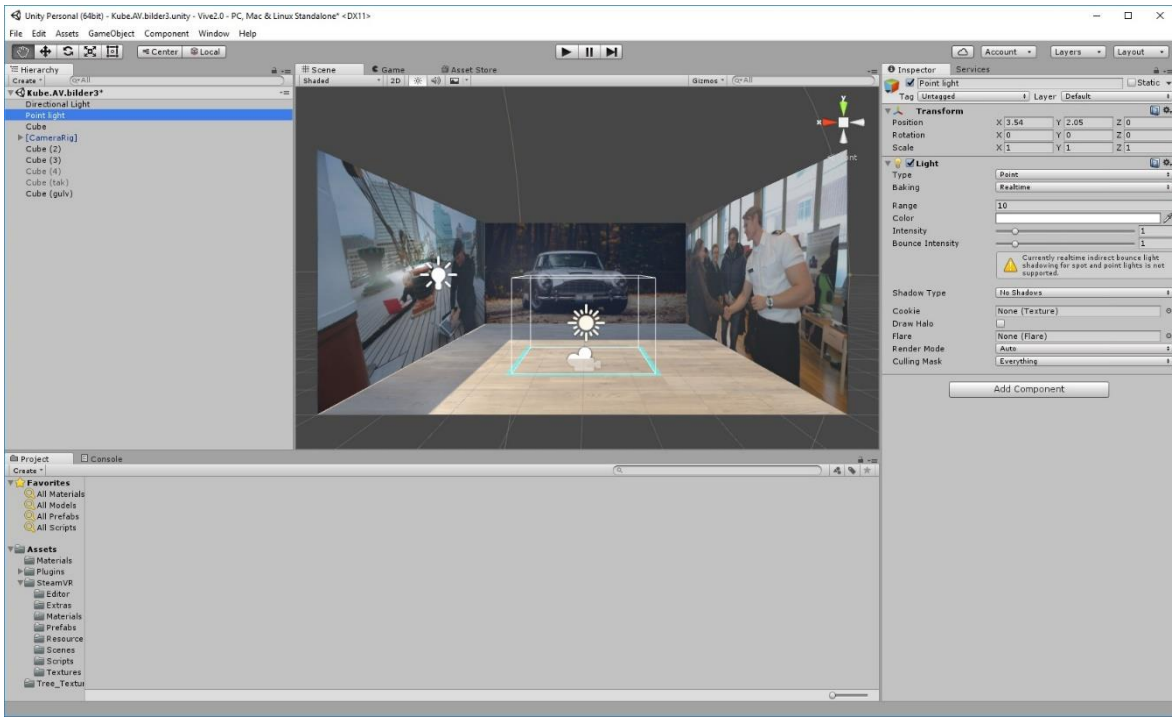


5.5.1 Figur Bildekube

Figur 5.5.1 viser hvordan vårt første bildegalleri ble designet.

Galleriet besto av 6 plan som var designet i en kubeformasjon. Disse planene hentet vi inn i det virtuelle rommet med akkurat samme fremgangsmåte som i delmål 2. I dette designet ble det kritisk å holde styr på posisjonen til hvert enkelt plan slik at kuben ble komplett. Vi brukte nyttige rotasjonserfaringer fra delmål 2. Det krevde god romforståelse å holde styr på hvilke akser som skulle roteres om hverandre.

Figur 5.5.2 viser en oversikt over hvordan kuben så ut fra innsiden (uten tak og framveg).



5.5.2 Figur Innside Bildekube

For å sørge for at vi hele tiden var kjent med hvordan Unity dekorerte bilder på de ulike planene lagde vi også her et programmeringsskript i C# slik at vi forstod prosessene. Hvert plan i kuben fikk hvert sitt skript. Det eneste vi endret på i de forskjellige skriptene var filnavnet på bildet som skriptet skulle hente opp fra harddisken. Ellers var det helt likt som skriptet i delmål 3. Disse skriptene ble ikke brukt videre i prosjektet, men de ga oss svært nyttig læring.

5.6 Delmål 5 Automatisk oppdatert bildeserie til lerret og galleri

En film er en rekke av fotografiske enkeltbilder, som vises i en rask rekkefølge. Dette skaper en illusjon av bevegelse som kalles phi-fenomenet. For å forstå hvordan en 360 graders film oppstår ønsket vi å prøve og lage vår egen lille film basert på egne enkeltbilder. For å forenkle delmålet bestemte vi oss for å lage et program som kun spilte av en bildeserie på 4 enkeltbilder ila. 16 sekunder. I prinsippet ble dette akkurat det samme som å lage en film med ekstremt lav «frames per second» ofte bedre kjent som fps. Et nytt bilde ble lastet inn hvert 4. sekund, som ga en fps lik 0.25, i motsetning til dagens moderne kamerateknologi som normalt sett ligger på rundt 30-60 fps.

Løsning ble å hente inn 4 bilder fra harddisken slik vi gjorde i de foregående delmålene, for så å lage en «Counter» funksjon i C#. Når datamaskinen hadde telt til et gitt tall, ble telle-funksjonen restartet, og et nytt bilde lastet opp og vist på lerretet.
Se vedlegg 10.2 Automatisk bildeserie.

5.7 Delmål 6 Hente inn live stream fra ulike kamera

Det var naturlig for oss etter delmål 5 å prøve og lage vår egen slags kinosal i det virtuelle rommet. Prinsippet ble akkurat det samme som i delmål 3, med unntak av at «texture» filen vi nå skulle dekorer planet med måtte bestå av rådataene fra et filmkamera i motsetning til våre egne enkeltbilder. På dette tidspunkt i prosjektet hadde vi lett tilgang til både webcam-funksjonen på en MacBook Pro og det integrerte kamera på VR-brillene.

Figur 5.7.1 viser de to kameraene vi valgte å bruke.

5.7.1 Figur Webkamera
Hentet fra www.engadget.com og www.gadgets.ndtv.com

I C# finnes det et eget bibliotek som henter inn rådata fra et webkamera. Når dette biblioteket kalles opp, søker C# automatisk etter tilgjengelige webkamera på maskinen og nummerer disse fra tallet 1 og oppover

- f.eks. WebCamTexure.deviceName=device [1].name;

Når brillene var koblet til MacBook maskinen hadde Unity tilgang på 2 forskjellige kamera. For å finne ut hvilke kamera som hadde hvilket nummer kjørte vi et enkelt forsøk der vi prøvde oss frem. Når kameranummeret var kjent for oss, kalte vi opp "Renderer" biblioteket

som dekorerte plan med rådata fra webkameraet. Ved å endre tallet i «Device[x]» kunne vi hente ut livestream fra begge kameraene i figur 5.7.1.
Se vedlegg 10.3 Rådata fra Webkamera.

5.8 Delmål 7 Hvordan presentere livestream fra 360 kameraet

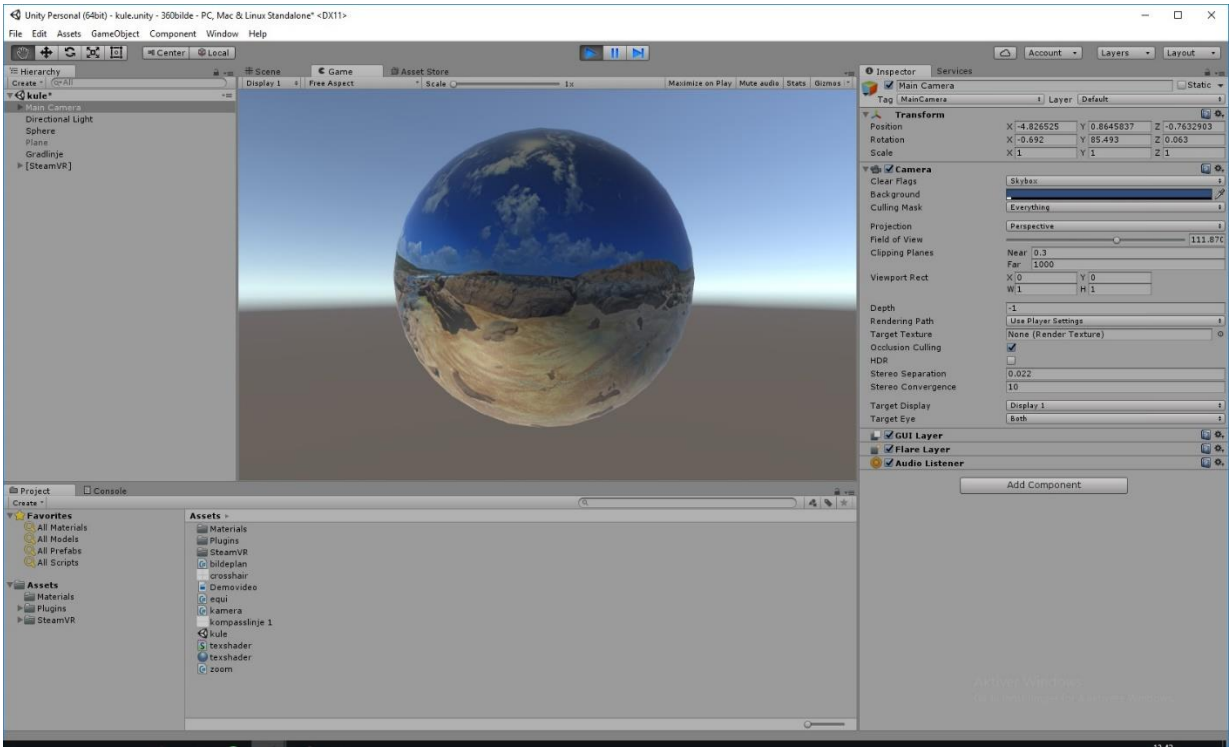
Ettersom poenget med hele produktet skulle være et 360 graders overvåkningssystem, var vi på dette tidspunkt nødt til å gå til innkjøp av et 360 kamera.
Henviser til 4.4 programvare, Ricoh Theta S.
Vi endte opp med å gå til innkjøp av Ricoh Theta S. Ved bruk av livestreamfunksjonen på dette kameraet ble videofilen sendt inn til maskinen som to 180 graders bilder.

Figur 5.8.1 viser rådataene maskinen mottok når kameraet sendte i livestream modus.

5.8.1 Figur Rådata 360 kamera
Hentet fra www.usatoday.com

For å presentere dette slik at operatøren av brillene fikk en følelse av selv å befinne seg i senter av bildet, la vi bildet som en «texture» på innsiden av overflateplanet til ei kule i det virtuelle rommet. Vi oppdaget raskt et viktig problem ved denne fremgangsmåten. Når vi i tidligere delmål hadde dekorert et plan med enten et enkeltbilde eller rådata fra et webkamera, hadde vi kun behøvd å dekorere utsiden av planet. Når vi da skulle legge rådataene fra 360 kameraet vårt inn på overflateplanet var det kun utsiden av kulen som ble dekorert når vi brukte kunnskapen fra tidligere delmål.

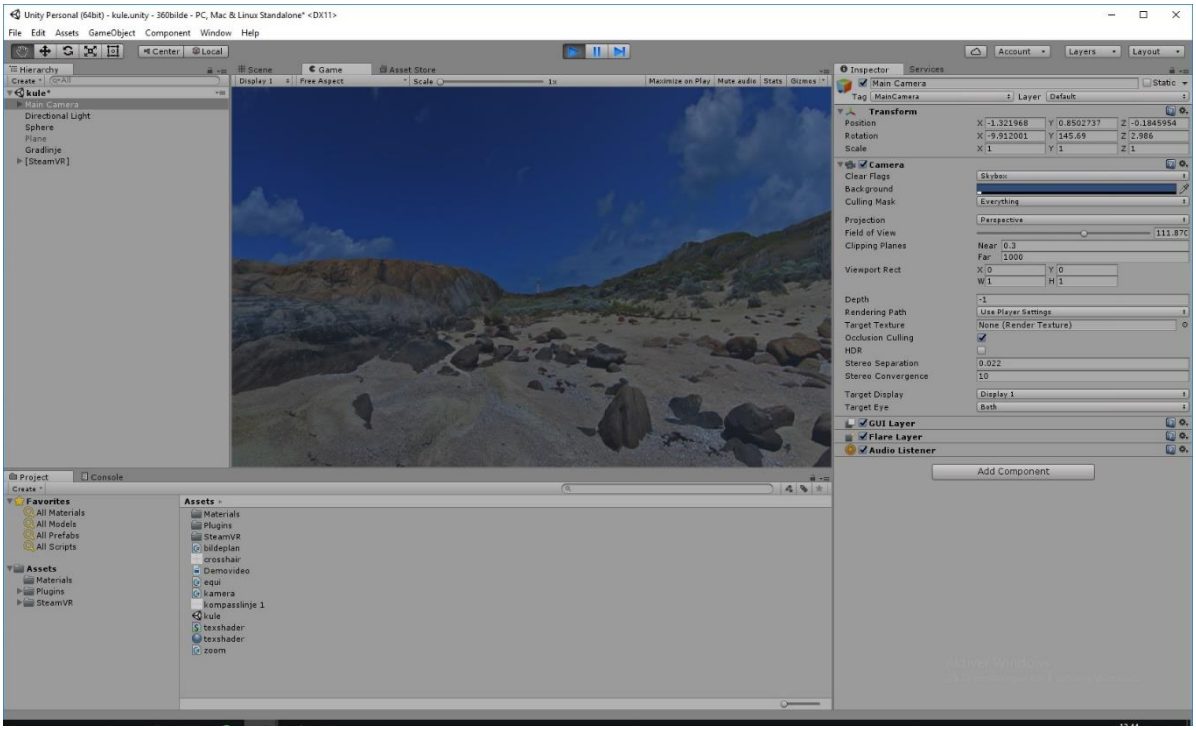
Figur 5.8.2 viser et eksempel på hvordan kulen vår så ut etter 360 bildet ble dekorert på. Som figuren viser ble kun utsiden av kulen dekorert.



5.8.2 Figur Virtuell kule Unity

Når vi plasserte VR-brillene i senter av kulen og kjørte programmet stod vi dermed inne i en kule uten noe innhold på innsiden av overflaten. Rådataene fra kamerat dekorerte utsiden, og vi så ingenting av det vi ønsket.

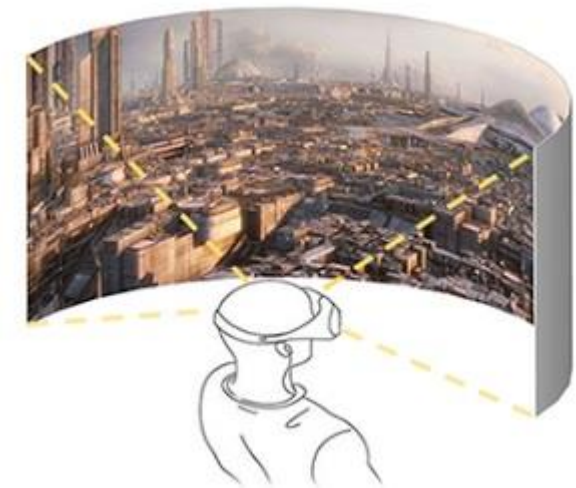
Løsningen ble å vrenge kuleplanet, slik at dekoren på utsiden ble innsiden. For å unngå at rådataene nå skulle bli presentert speilvendt inne i kulen, måtte vi speilvende utsiden. Dette gjorde at bildet ble korrekt fremstilt når overflateplanet ble vrengt.



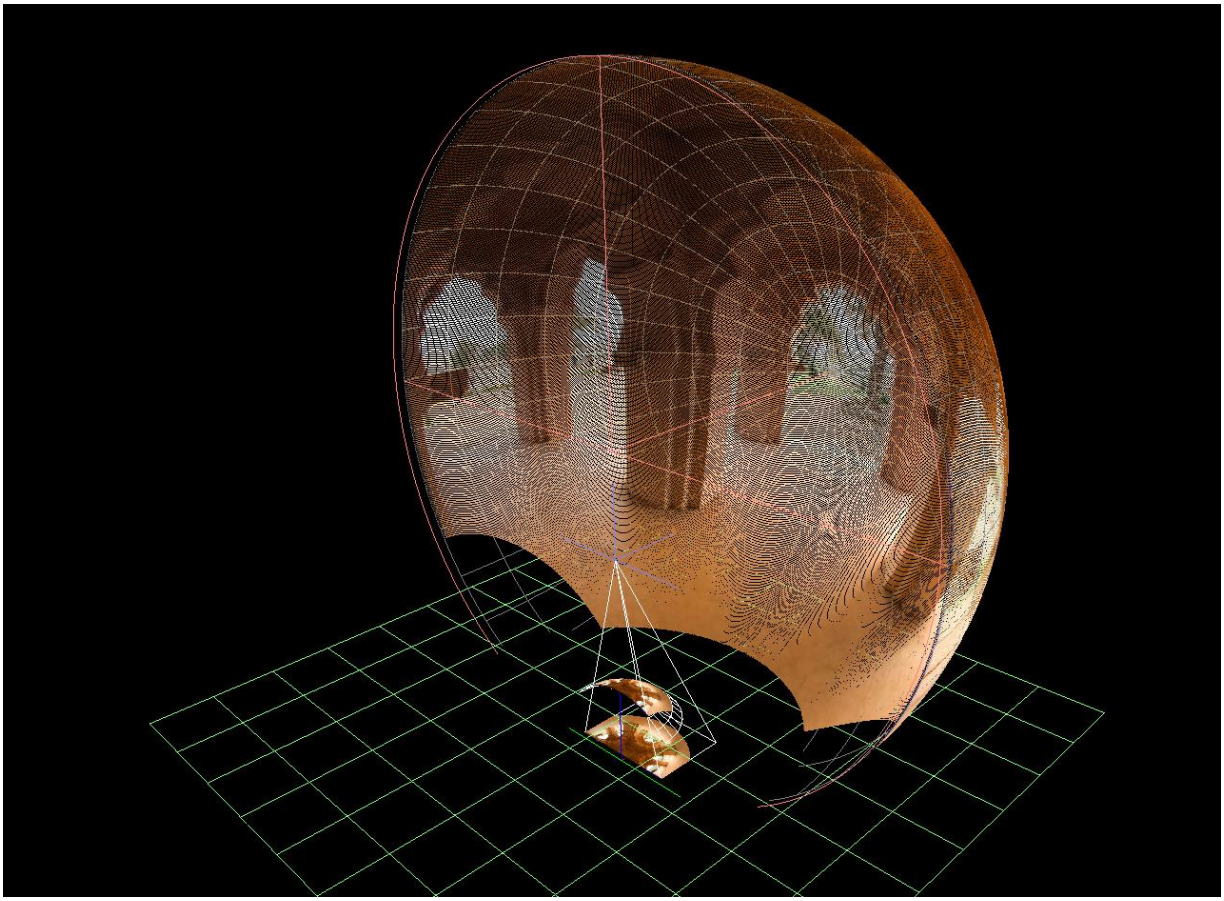
5.8.3 Figur Innside Virtuell kule Unity

Figur 5.8.3 viser et utsnitt av hvordan opplevelsen av et ekvirektangulært bilde så ut på innsiden av kuleoverflaten. Det som vises i figuren er ikke rådata fra Webkameraet, men et demonstrasjonsbilde som vi her har lagt inn for å vise opplevelsen av 360 bilde i Unity.

Figur 5.8.4 og 5.8.5 demonstrer hvordan rådataen fra et webkamera blir presentert på innsiden av ei kule og på den måten demonstrer prosessen som gjøres når 360 graders videofilmer skapes.



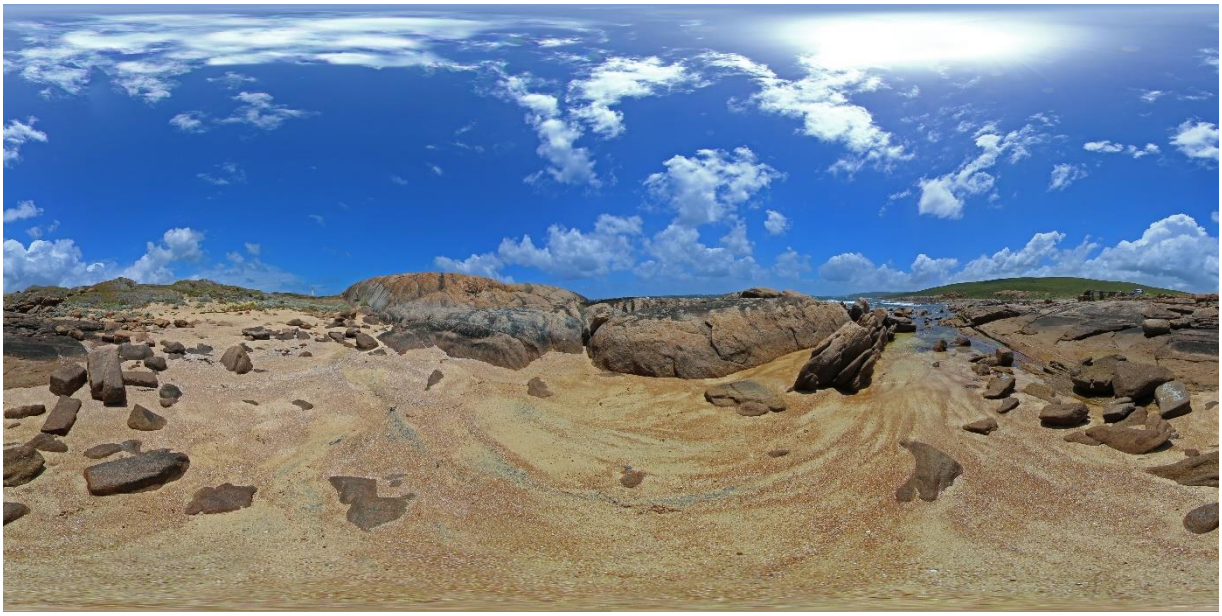
5.8.4 Figur Presentasjon 360 video
Hentet fra www.360rumors.blogspot.com



5.8.5 Figur Demonstrasjon 360 video
Hentet fra www.paulbourke.net

For å kunne dekke kuleflaten med kamerabildene som vist i figur 5.8.4 trenger man en mapping mellom bildene og kuleflaten. For ekvirektangulære bilder, eksemplifisert i figur 5.8.6, gjør Unity denne mappingen automatisk. Utfordringen var imidlertid her at Ricoh-kameraet produserer fisheye-bilder, som vist i figur 5.8.1. Dette medførte at vi måtte lage mappingen fra bildekoordinater til kuleflaten selv.

Grunnlaget for mappingen er teorien beskrevet i avsnitt 3.3, Fisheyemodellen. For å få konverteringen effektivt nok til å fungere i sanntid måtte vi implementere den som en shader som kjører på grafikkortet. Selve implementasjonen ligger i SkyboxShader. I shaderen blir `frag()`-funksjonen kalt for hvert piksel som tegnes opp. Vi finner hvilket punkt på kuleflaten dette pikselet peker mot (`i.texcoord`) og konverterer dette til kulekoordinater. Så bruker vi formlene for fisheyemodellen og finner det normaliserte bildepunktet `p_`. Til slutt gjenstår litt småjustering for å treffe riktig linse i råbildet fra kameraet.



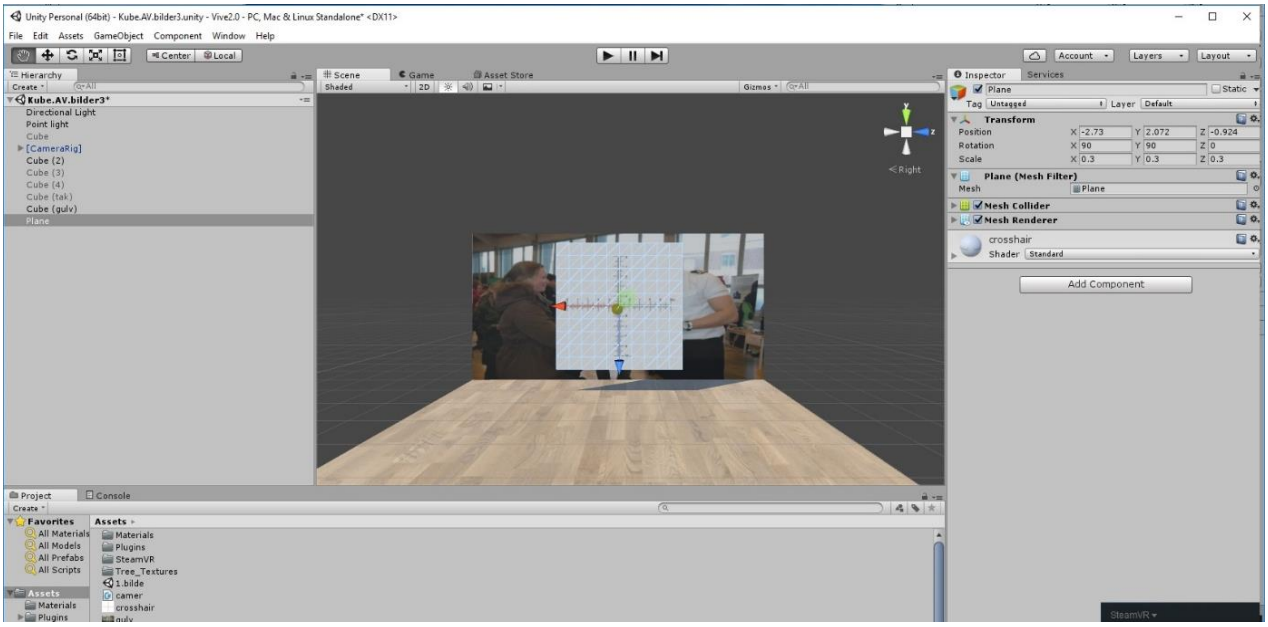
5.8.6 Figur Ekvirektangulært bilde
Hentet fra www.paulbourke.net

5.9 Delmål 8 Konstruksjon av et siktekors i senter av synsfeltet i brillene

Som vist i delmål 2, kan man i Unity konstruere ulike geometriske figurer i samme virtuelle bildet. Vi har frem til nå designet et program med ei kule som fremstiller rådataene fra et 360 kamera på en måte som gir operatøren en følelse av at han befinner seg i senter av bildet. Nå ønsket vi å lage et siktemiddel slik at operatøren kunne få et referansepunkt å forholde seg til i senter av det virtuelle bildet.

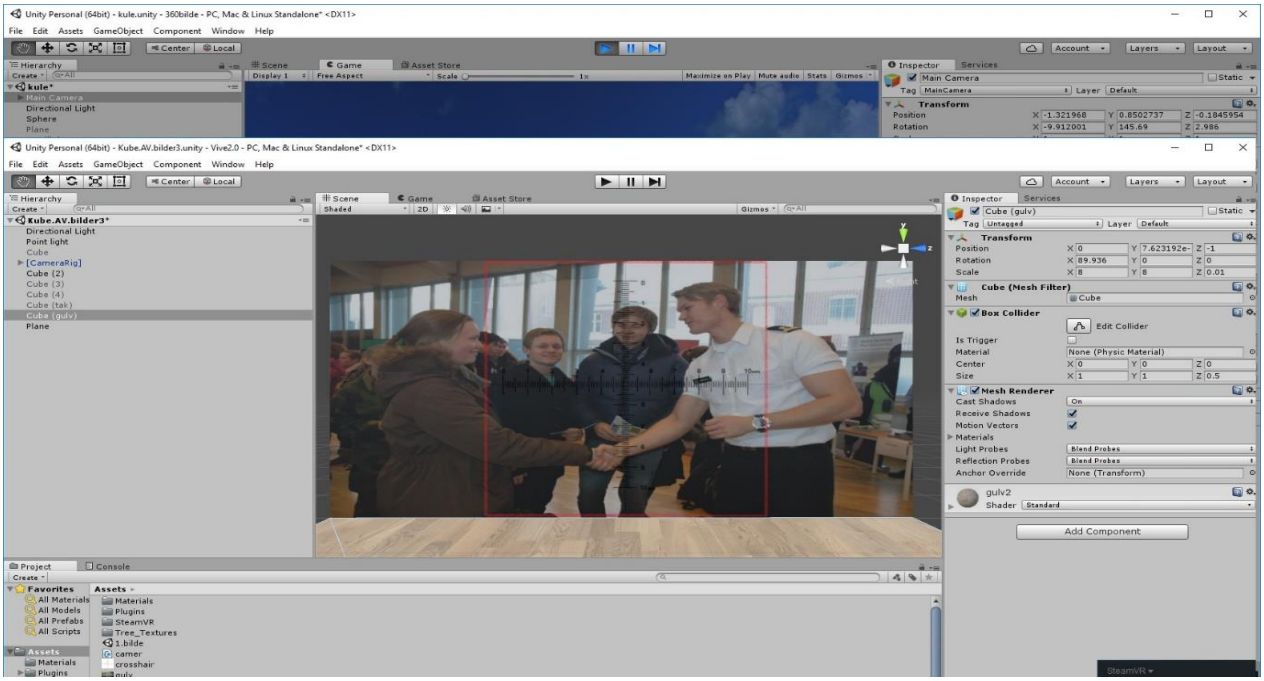
Får å løse dette tilførte vi et vertikalt plan som vi dekorert med et ønsket siktekors. Dette planet la vi mellom VR-brillenes posisjon i det virtuelle rommet og kuleoverflaten.

Figur 5.9.1 på neste side viser en forenklet demonstrasjon vi designet for å forklare prosessen.



5.9.1 Figur Planoppbygging siktekors

Eksempelet viser lerretet vi designet i delmål 3. Foran veggplanet har vi konstruert ett nytt plan som vi har dekorert med det siktekorset vi ønsket å benytte oss av. For å unngå at siktekorset ble en heldekkende firkant med hvit bakgrunn, brukte vi en funksjon i Unity som gjør det mulig å endre hvordan planet skal beklès av bildefilen. Alle piksler i bildet som ikke har fargen svart, blir gitt fargen til planet på baksiden, dermed er det kun siktekorset som synes i det virtuelle bildet.



5.9.2 Figur Demo siktekors

Figur 5.9.2 på forrige side viser resultatet etter at denne innstillingen ble gjort.

Siktekorsplanet er rammet inn i fargen rødt, og vi ser at det nå kun er siktekorset som syntes i det virtuelle bildet.
Får å oppnå denne effekten endret vi «shader» innstillingen. Endrer man shader fra standard til partical-multiply fremstår alt annet enn svarte piksler i bilde som gjennomsiktig.

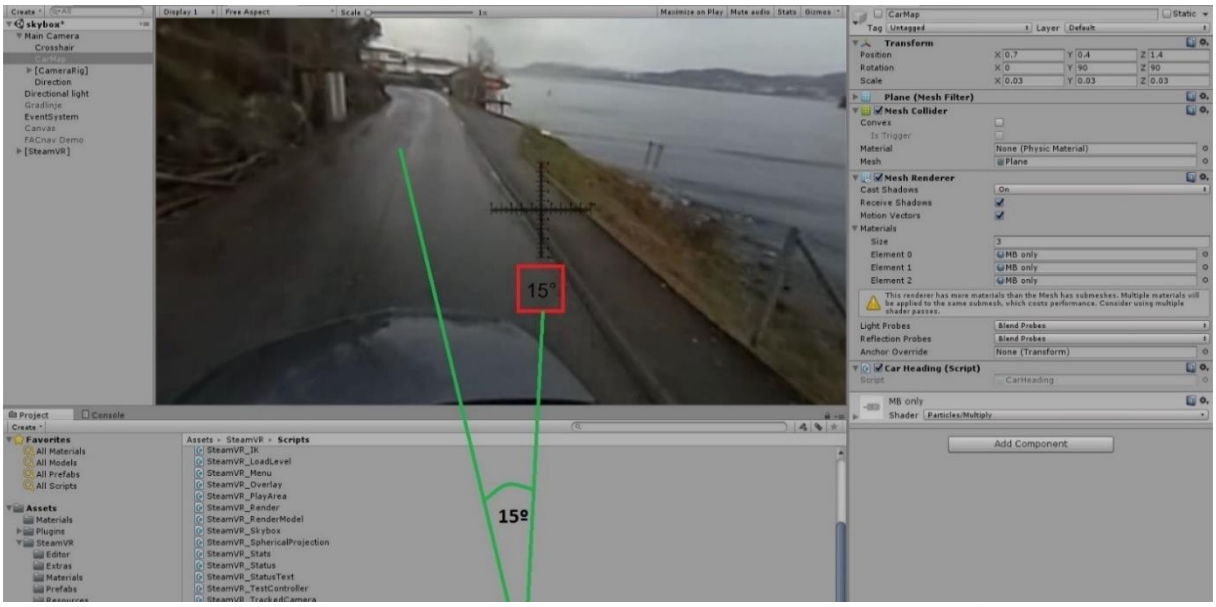
Akkurat samme prosessen som eksempelet 5.9.1 og 5.9.2 viser, gjorde vi også i kulen vi hadde designet tidligere. Det kunne kreve forholdsvis mye arbeid å få dette siktekorsplanet til å legge seg perfekt i mellom VR-brillenes posisjon og kuleoverflaten inne i kulen. For å gjøre det lettere for oss å konstruere flere plan inne i det virtuelle bildet flyttet vi presentasjonen av rådataene bort fra den geometriske kulen og inn på horisonten i Unity.
Hver gang Unity startes opp har allerede programmet laget et default virtuelt univers for deg. Dette universet inneholder kun en blå horisont langt der bak i det fjerne. Figur 5.3.2 gir deg en god demonstrasjon på hvordan denne horisonten ser ut. Istedenfor at Unity selv designet denne horisonten som en himmel, la vi våre egne data fra kameraet inn som horisont. Denne horisonten er egentlig bare ei stor kule som VR-brillene blir plassert i midten av når programmet starter opp. I Unity kalles denne horisonten for «skybox», og ved å legge vår rådata inn som default skybox shader ble det mye lettere for oss å arbeide med andre plan i det samme rommet. Nå lå hele 360 filmen presentert langt foran oss i det virtuelle rommet og det kom dermed ikke i konflikt med nye plan som vi designet. Dybdeeffekten ble også mer realistisk da 360 filmen ble presentert på horisonten. Det ga en virkelig 3D-følelse.
Se vedlegg 10.4 Skybox texture.

5.10 Delmål 9 Hvordan orientere seg i VR-rommet?

For å gjøre det intuitivt for operatøren å orientere seg i det virtuelle bildet, konstruerte vi et head up display (HUD) som viste oversikt over brillenes rotasjon om y-planet. Denne funksjonen skulle også gjøre det mulig for operatøren å raskt fortelle i hvilke retninger han gjør eventuelle måloppdagelser. I første omgang lagde vi denne orientering med utgangspunkt i himmelretningene. Rotasjonen går fra 0 til 360 grader, der;

- 0° refereres til himmelretningen nord
- 90° rett øst
- 180° rett sør
- 270° rett vest

Figur 5.10.1 viser en oversikt over hvordan gradene ble regnet ut i programmet. Her har vi tatt utgangspunkt i at kjøretøyets fartsretning og himmelretningen nord er like. I fartsretning legger vi bilens rett frem orientering. Denne retningen oppleves når fører sitter i framsete av kjøretøyet og ser rett ut gjennom frontvinduet.



5.10.1 Figur Referansepunkt synsfelt

For å designe denne funksjonen var vi avhengig av å finne ut brillenes rotasjon om y-planet. Henviser til Teoridel 3.9 Light house tracking Technology.

Quaternion **Euler**(float **x**, float **y**, float **z**); er en funksjon i skriptet som henter ut rotasjonen i grader om x, y og z aksene, tatt ut i fra Unity sin DefaultRotation. Denne funksjonen brukte vi i programmet vårt slik at vi fikk ut gradene brillene hadde rotert om y-aksen. For å styre referansepunktet Unity bruker for å regne ut rotasjonen la vi inn en funksjon som gjorde at vi selv kunne stille inn hvilken retning referansepunktet skulle være ved å trykke tastet "n". Det nye default-referansepunktet til brillene ble da den retningen som senteret i brillen var rettet mot når operatøren trykket tasten n.

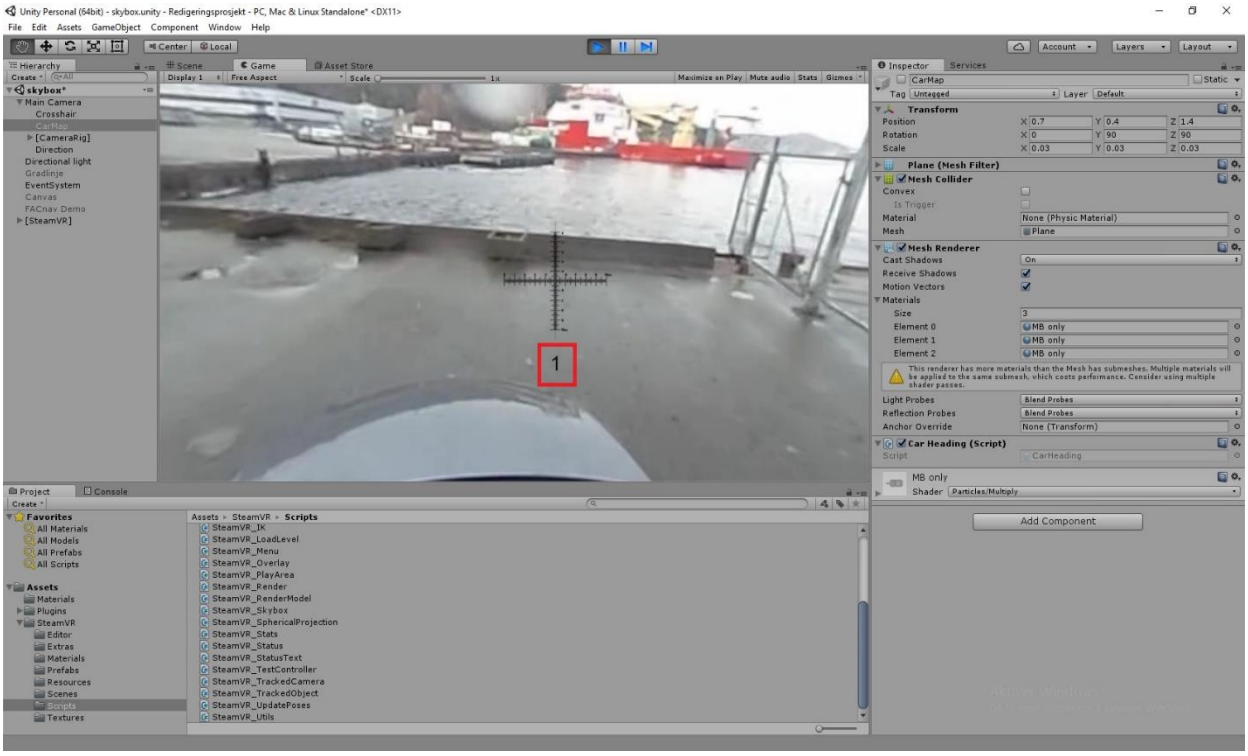
Når denne funksjonen skal tas i bruk i den virkelige verden er det viktig at dette referansepunktet til Unity blir kalibrert ved hjelp av kjøretøyets kompass slik at det stemmer med virkeligheten.

I diskusjon og testing med operatørene fra MJK fikk vi tilbakemelding på at de ikke nødvendigvis orienterte seg i forhold til himmelretningene ved forflytning i kjøretøy. De

eneste situasjonene operatørene trengte å oppgi retning, på f.eks måloppdagelse, i enheten grader, var når de skulle styre ildstøtte som kom fra luften. De ønsket at vi på en eller annen måte lagde funksjonen slik at den orienterte seg etter kjøretøyets fartsretning, der rotasjonen var oppgitt etter en klokke.

- Rett frem henviser til kl. 12
- Rett høyre kl. 3
- Rett bakover kl. 6
- Rett venstre kl. 9

Figur 5.10.2 viser hvordan klokkemetoden blir fremstilt for operatøren.



5.10.2 Figur Klokkemetoden

Dette løste vi ved å lage følgende matematisk omregningsformel fra grader til klokke:
(heading / 360 * 12 + 11.5) % 12 + 1;
Vi henter ut headingen til brillene, den får vi i grader. Default rotation i brillene representere 0 grader. Deretter transformerer vi enheten fra grader til timer, ved å dele 360 på 30, og runde ned til nærmeste heltall (da vi kun ønsket at displayet skulle vise hele timer). For å sørge for at klokkeintervallet fra 11.30 til 12.30 vises som 12 i displayet måtte vi forskyve hele

systemet med en halvtime. Dette gjorde at intervallene videre rundt på urskiven ble korrekt. Alle klokkeslett ble definert ved intervallet 30 min før til 30 min etter.

.

Operatøren kan ved å veksle mellom tastene "j" og "k" bestemme om han ønsker å orientere seg etter himmelretningene eller klokkemetoden. I vårt program styres nå begge innstillingene av ett og samme referansepunkt, dette må endres på når produktet blir implementert på et kjøretøy. Da er det viktig at det skilles mellom de to referansepunktene, slik at grader kalibrers etter et kompass, mens klokkemetoden kalibres etter kjøretøyets fartsretning rett fremover.
Se Vedlegg 10.5 Orientering virtuelt rom.

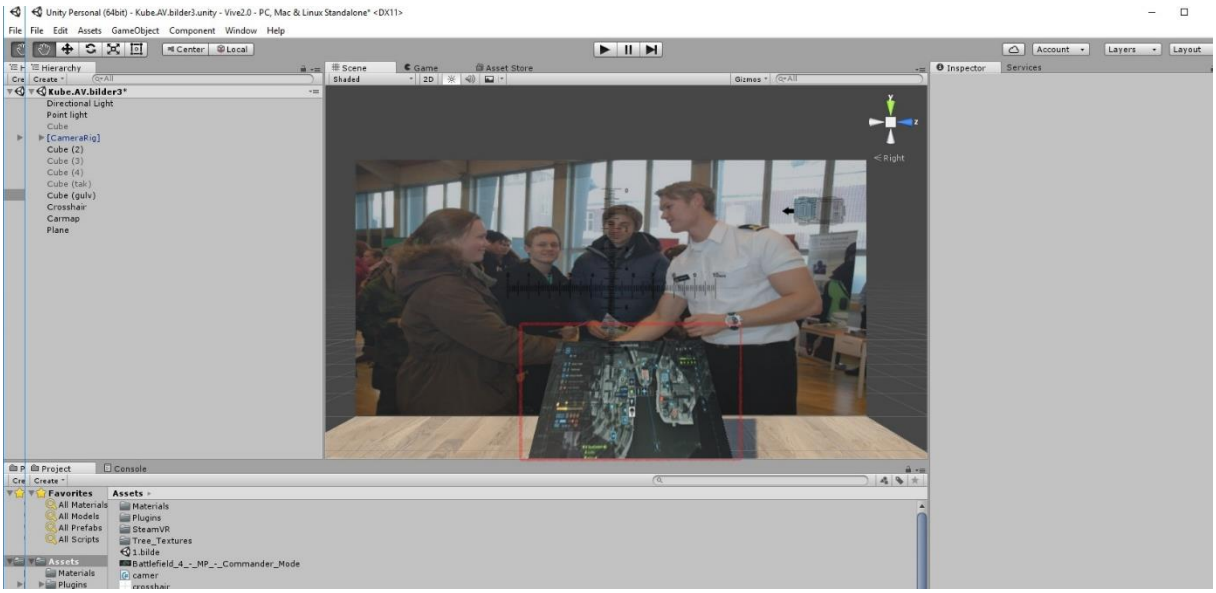
5.11 Delmål 10 Produktet må ha form for verdensorientering (kart)

Vi ønsket at produktet skulle inneholde en måte å holde oversikt over kjøretøyets posisjon ut over det som øye kunne oppfatte gjennom kamera. Den første tanken var å legge ett vertikalt plan oppe i ene hjørne av synsfeltet som skulle inneholde et kart over området kjøretøyet befant seg i. Men i samråd med Marinejegerkommandoen ble vi enige om å heller legge dette kartet som et horisontalt plan over kjøretøyets egen takkonstruksjon. Operatøren har liten interesse av å observere taket på sitt eget kjøretøy ute i operasjoner, derfor var det bedre å bruke denne plassen istedenfor at et slikt kartplan skulle gå på bekostningen av operatøren synsfelt oppe i et av hjørnene. Både operatører fra marinejegerkommandoen og kystjegerkommandoen var tydelig på viktigheten av det engelske uttrykket «Less is more» når de ga sine instrukser for brukergrensesnittet.
Kartet ble lagt inn på akkurat samme måte som vi har dekorert tidligere plan. For å gjøre det mest mulig intuitivt for operatøren å orientere seg i kartet lagde vi et skript som sørget for at kartplanet alltid orienterte seg etter synsfeltet til operatøren. Dette gjorde vi ved å hente ut brillenes posisjon på akkurat samme måte som i delmål 9. For at kartplanet ikke skulle følge med operatøren når han roterte hode om x-aksen (opp-ned bevegelse) satte vi denne verdien til å være konstant. Under ser du en oversikt over de restriksjoner vi la inn for planets rotasjoner.

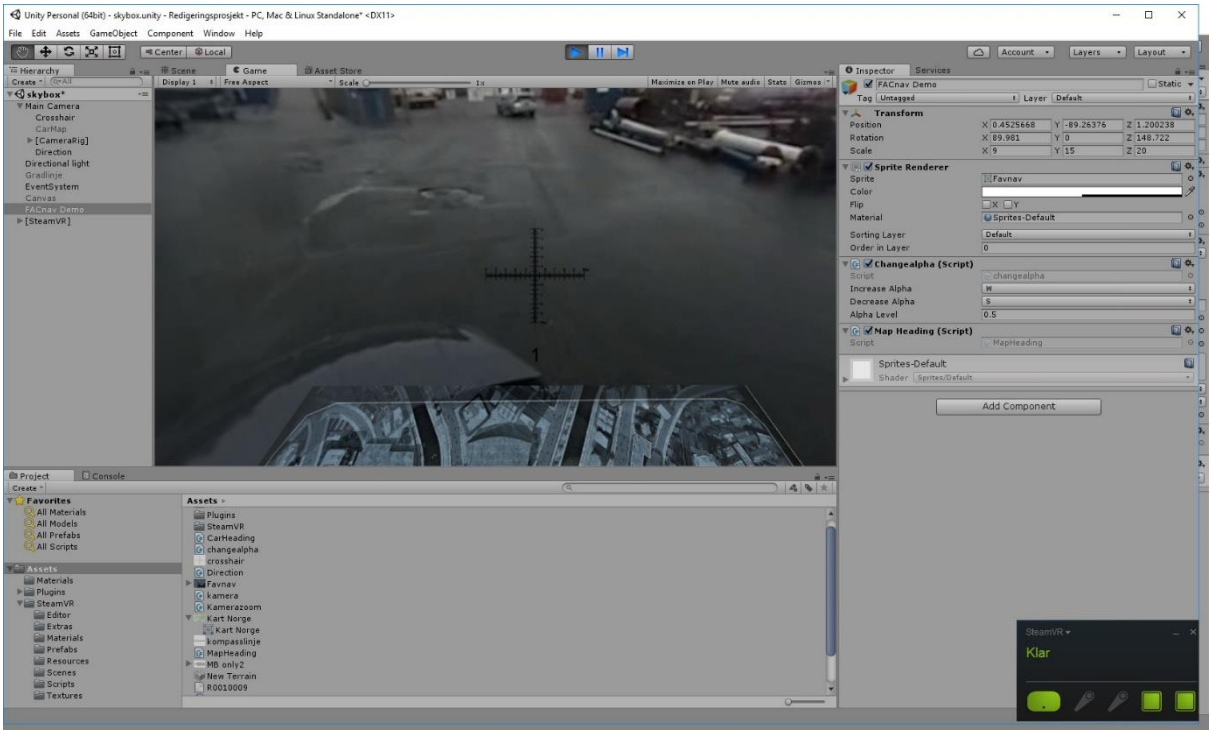
```
gameObject.transform.localEulerAngles = new Vector3(90, rot.eulerAngles.y, rot.eulerAngles.z);
gameObject.transform.position = pos + (new Vector3(0, -90, 0));
```

Se vedlegg 10.6 kartorientering.

Figur 5.11.1 demonstrer hvordan kartplanet legges inn i lerret eksempelet fra tidligere demonstrasjoner, mens figur 5.11.2 viser hvordan det ble seende ut i det virkelige produktet.



5.11.1 Figur Oppbygging kart



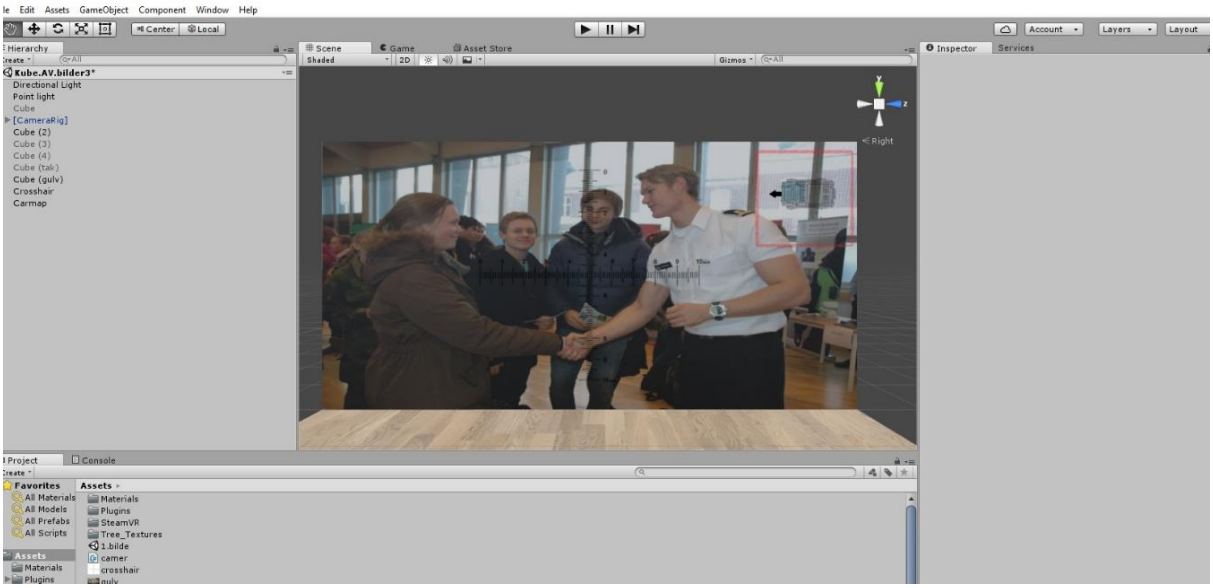
5.11.2 Figur Demo kart

For å gi operatøren muligheten til å justere opp og ned lysstyrken på kartet ettersom lysnivå ute forandrer seg gjennom dagen, lagde vi en tilleggsfunksjon som ga operatøren mulighet til å endre disse innstillingene ved å trykke på tastene «w» og «s». Denne funksjonen utføres av

Unity sin egen "alpha channel" funksjon. I korte trekk sørger funksjonen for at det til enhver tid blir lagret to bilder i ett Alpha Channel minne. Et av bildene er ditt originale bilde (i dette tilfelle kartet), mens det andre bilde er av bakgrunnen. Gjennomsiktighetsfølelsen skapes gjennom at du selv styrer pikselfordelingen mellom disse to bildene. Vil du øke gjennomsiktigheten i bildet, økes antall piksler fra det opprinnelige bakgrunnsbildet ditt i totalbildet, og øyet oppfatter at bildet ser mer gjennomsiktig ut.
Se vedlegg 10.7 Gjennomsiktighet.

5.12 Delmål 11 Evnen til å vite kjøretøyets retning

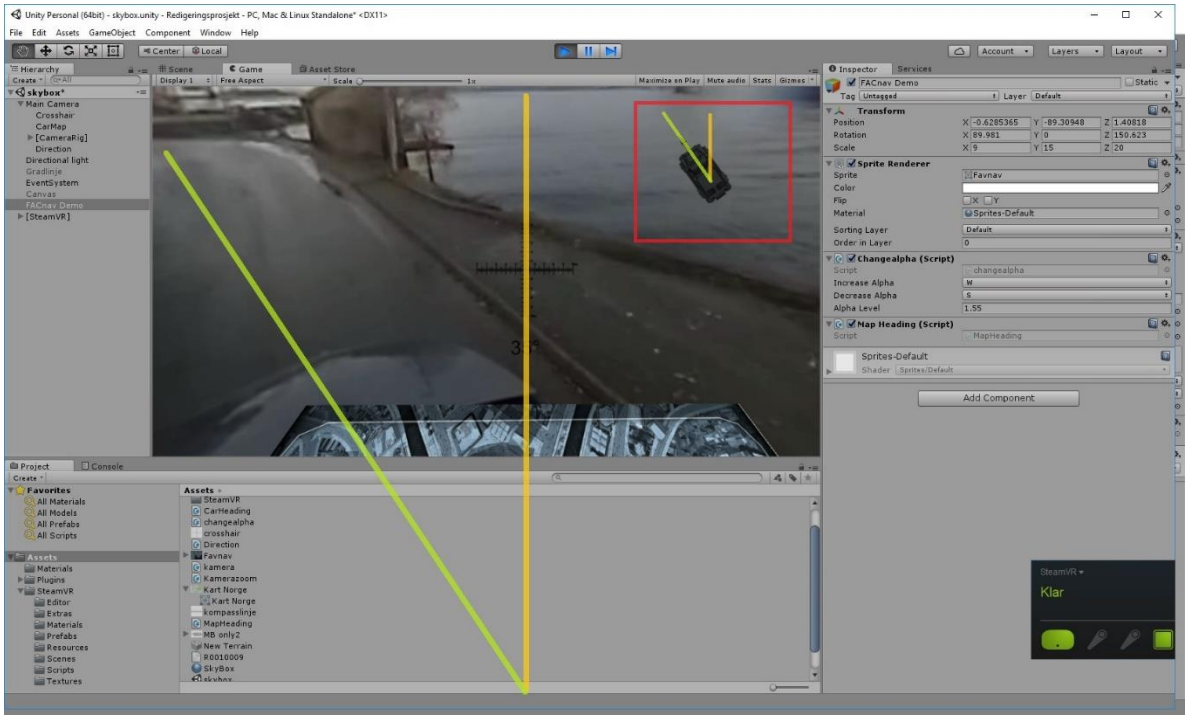
For å raskt orientere seg i det virtuelle bildet, så vi det som hensiktsmessig at operatøren til enhver tid hadde kontroll på hvilken retning kjøretøyet pekte. Ved hjelp av en slik modell håper vi at hjernen lures til å akseptere at det den observerer gjennom brillene er virkeligheten. Hvis operatørens hjerne til enhver tid har kontroll på hva som er fartsretningen på kjøretøyet, vil bevegelsen inne i kjøretøyet forhåpentligvis føles mer naturlig og logiske for kroppen, og vil minimere sjansen for kvalme og svimmelhet. Samtidig er det praktisk for operatøren å ha et visuelt bilde på hvordan kjøretøyet er orientert. I prinsippet vil klokkemetoden på siktekorset i delmål 9 også gi deg informasjon om hvor du befinner deg i forhold til kjøretøyets fartsretning, men et visuelt bilde vil gi operatøren muligheten til å få en utrolig rask og intuitiv grovorientering. Får å få inn en slik kjøretøymodell brukte vi akkurat samme fremgangsmåte som når vi lagde siktekorset i delmål 8.
Figur 5.12.1 viser hvordan dette planet ble seende ut i eksempel demonstrasjonen.



5.12.1 Figur Kjøretøysorientering

For at planet skulle angi kjøretøyets fartsretning var vi nødt til å sørge for at planet snudde seg ettersom operatøren roterte synsfeltet. Akkurat som klokkemetoden hentet ut referansepunktet sitt fra Unity sin "defaultRotation" sørget vi for at dette planet alltid var orientert mot "defaultRotation". For at denne funksjonen skulle fungere som planlagt måtte «defaultRotation" alltid være kalibrert slik at den var identisk med kjøretøyets virkelige fartsretning.
Se vedlegg 10.8 Kjøretøysorientering.

Figur 5.12.2 viser hvordan modellen roterer seg etter kjøretøyets virkelige posisjon i brillene. Senter i synsfeltet er merket med gult og kjøretøyets fartsretning med grønt.



5.12.2 Figur Referansepunkt kjøretøysorientering

5.13 Delmål 12 Endelig produkt

Avslutningsvis legger vi ved noen bilder av hvordan det virtuelle bilde for operatøren ble seende ut fra forskjellige vinkler, og med en blanding av grader og klokkemetode som orienteringsverktøy. Bildene er hentet fra TEST 3.

5.13.1 Figur Prototype 1

5.13.2 Figur Prototype 2

5.13.3 Figur Prototype 3

6 TESTING

I denne delen av oppgaven vil vi gå igjennom ulike tester som er blitt kjørt av programmet. I vårt tilfelle har prosjektet kontinuerlig blitt testet fra start til slutt, og vi har laget egne delprosjekter for alle de 11 delmålene vi satte oss i utviklingsfasen. For å kontrollere at hver funksjon fungerte korrekt, og at resultatet ble slik vi ønsket, testet vi kontinuerlig underveis. Disse testene har vi valgt å ikke belyse under punktet testing. Vi har derimot valgt ut 3 signifikante hendelser som hadde stor betydning for utformingen av produktet.

6.1 TEST 1: Bevegelser i 1:1

I begynnelsen av utviklingen arbeidet vi på en MacBook Pro. Denne maskinen hadde i utgangspunktet ikke de kapasiteter verken Unity eller VR-brillene krevde for å kjøre optimalt. Særlig grafikkortet i Mac maskinen hadde problemer med å kjøre brillene. Dette gjorde at vi under testing i begynnelsen opplevde mye «lagging» og forsinkelser i bildet. Det var liten grad av virkelighetsfølelse og man kunne etter kort tid føle seg kvalm og svimmel. For å sjekke at produktet vårt, med tilstrekkelig datakapasitet, hadde potensial for å fungere slik vi hadde tenkt, utførte vi en produkttest på en av Forsvarets forskningsinstitutt (FFI) sine maskiner.

Med testen ønsket vi å sjekke om denne VR teknologien virkelig kunne gi en reell følelse av virkeligheten. Ut ifra samtaler med Kongsberg ProTech System i begynnelsesfasen av prosjektet, var vi kjent med problemene de hadde rundt kvalme og desorientering når de har prøvd å lage tilsvarende produkt. Vi så for oss at hvis operatørene opplevde en 1:1 bevegelse i det virtuelle rommet ville mye av problematikken rundt kvalme og svimmelhet forsvinne. Resultat var over all forventning, selv for vår veileder ved FFI. Brillene fungerte med svært liten, nærmest ingen forsinkelser, tregheter eller unaturlig bevegelser. Følelsen av å benytte brillene og bevege hodet rundt i det i virtuelle rommet var så godt som 1:1. Testen ga oss masse motivasjon for videre utvikling, og hadde stor betydning for at denne idéen faktisk kunne bli virkelighet.

6.2 TEST 2: Marinejegerkommandoen og Kystjegerkommandoen

Allerede fra begynnelsen av prosjektet har vi spilt på lag med operatører fra MJK og KJK. Vår grunntanke handlet om å skape et teknologisk hjelpemiddel, utviklet av ingeniører og brukere sammen. På denne måten kunne operatørene komme med innspill på hvordan de

ønsker at produktet skulle fungere og dermed sørge for at teknologien ble tilpasset deres operasjonsmønster, og ikke omvendt. Vi så potensialet av å ha disse operatørene tilgjengelig i hverdagen, og ønsket å bruke dette til det beste for begge parter. Forholdet til MJK og KJK kan ses på mer som et kontinuerlig samarbeid enn en enkelt test, og det besto av flere diskusjoner og møter. Likevel ønsker vi her å ta frem et spesielt møte som la grunnlaget for hele brukergrensesnittet vårt.

Etter vi var ferdig med delmål 9 og var i stand til å fremstille en livestream av omgivelsene inne i brillene inviterte vi MJK og KJK på første virkelig test. Det ble en lang diskusjon om potensial og viktige funksjoner som operatørene ønsket å ha i brillene. Ut ifra denne testen kom vi frem til et viktig moment vi tok med oss gjennom oppgaven. Operatørene ønsket at vi hele tiden hadde det engelske prinsippet «Less is more» i bakhode når vi designet noe i grensesnittet. De hadde erfart fra tidligere teknologi at mye tilleggsfunksjoner og fancy informasjon ble festet på utstyret de brukte, uten at det egentlige behovet var til stede. Dette skapte ofte mer støy enn det ga tilbake i gevinst. Dette ble viktig for oss å unngå. De ga oss i tillegg konkrete tilbakemeldinger på hvordan de ønsket at flere av de ulike funksjonene skulle fungere. Spesielt viktig for dem var det at orienteringsfunksjonen opprinnelig tok utgangspunkt i klokkemetoden, og ikke grader slik vi først hadde trodd de ønsket. MJK var svært tydelig på at eventuelle display som gikk på bekostning av operatørens observasjonsområde var svært lite ønskelig. Derfor ble vi enig om å legge FACnav planet ned i operatørens synsfelt slik at det ikke var til hinder for observasjon. Alt i alt formet denne testen svært mye for hvilken retning prosjektet gikk videre.

6.3 TEST 3: Livetest fra kjøretøy

Når all programutvikling var ferdig, ønsket vi å teste hvordan produktet fungerte ute i den virkelige verden. Frem til dette tidspunkt hadde vi kun sittet inne på et arbeidsrom og gjennomført alle tester og demonstrasjoner.

Vi rigget opp ett stativ som gikk opp igjennom soltaket på et disponibelt kjøretøy, og kjørte en runde rundt i nærområdet mens kameraet filmet fra taket i den hensikt å simulere hvordan produktet faktisk vil fungere ut i felten.

Resultatet var overaskende. Både oss, MJK og KJK var meget overasket over hvor virkelighetsnært det føltes å se gjennom brillene. Når man hadde på brillene var det nærmest som en selv satt på toppen av kjøretøyet og observerte. Dette var slik vi hadde sett for oss i begynnelsen. Denne testen ble akkurat den bekreftelsen vi trengte for å bekrefte at dette

produktet virkelig er noe for fremtiden. Med et bedre 360 kamera, tror vi at dette produktet kan bli til stor hjelp for soldater ute i felt.

6.4 Test 4: Bildekvalitet, Briller eller kamera?

I retrospekt har vi gjort oss opp noen tanker om tester vi helst skulle ha utført, men som vi ikke har hatt tid til. Blant annet skulle vi gjerne sett nærmere på hvilke av produktdelene våre som setter en begrensning for den bildekvaliteten operatøren opplever i brillene. Vi har testet brillene med ekvirektangulære bilder med høy oppløsning, hentet fra google. Vi fikk en bekreftelse på at brillene håndterte høy oppløsning på rådata uten problemer. Dog hadde det vært interessant å teste produktet med et bedre 360 kamera, eksempelvis det nye Gopro Omni 360 kameraet som nettopp er lansert. Det hadde vært meget interessant og sett hva dagens nyeste kamerateknologi kunne tilført produktet vårt.

7 Konklusjon

Problemstillingen vi stilte oss selv innledningsvis i oppgaven var:
"Hvordan kan vi, som teknikere, ved bruk av teknologiske hjelpemidler, bidra til at våre medsoldater opprettholder helhetsoversikten, uten at de selv eksponeres."

I vårt arbeid med denne oppgaven har i løpet av perioden lært hvordan man programmerer og skaper en virtuell plattform designet for VR-briller. Vi har benyttet rådata fra et 360 kamera som vi videre har bekledd atmosfæren med i et virtuelt rom som vi har skapt. Dette har igjen gitt oss en mye større forståelse og kunnskap om bildeprojeksjon, rendering og C# programmering. Vi har i oppgaven kommet borti betraktelig mer faglig innhold knyttet opp mot Sjøkrigsskolen enn det vi forventet før vi startet arbeidet. Vi oppdaget mye underveis, eksempelvis mengden av matriseregning og Kalmanfilter. Vi har konstruert en plattform hvor man med VR-briller kan se liveoverføring av et 360 bilde, i sanntid. Vi har lagt inn et brukergrensesnitt i programmet som består av siktekors, kjøretøysretning, klokke og kompass orientering, samt kart. Alt for at operatøren skal ha tilgang på nyttig informasjon, presentert direkte i synsfeltet. Irrelevant informasjon skal ikke stjele fokus. Brillene skal kun vise kritisk informasjon.

Vi har igjennom prosessen samarbeidet med operatører fra Marinejegerkommandoen og Kystjegerkommandoen. Dette for å kartlegge hva brukere som skal benytte utstyret mener er kritisk eller irrelevant informasjon.

Prosessen med å utvikle denne sammenkoblingen mellom liveoverføring av 360 bilde og VR-briller har vært lang og krevende. Det har vært mye reising frem og tilbake mellom vår daglige base, Sjøkrigsskolen og Forsvarets Forskningsinstitutt på Kjeller. Det har vært mye frustrasjon over småfeil og manglende forståelse. Denne frustrasjonen har dog som regel gått over til glede etter gode dialoger med vår veileder på FFI. For å konkludere har vi som teknikere gjennom denne oppgaven skapt en plattform hvor våre medsoldater kan opprettholde helhetsoversikten, uten at de selv eksponeres. Vi er svært fornøyd med resultatet.

7.1 Kommentar fra FFI-veileder

Rundt om i verden jobbes det for tiden mye med 360-graders situasjonsforståelse basert på ulike sensortyper, både som operatørstøtte og for autonome systemer. Vi ser at hovedutfordringen er å nyttiggjøre seg av all informasjonen som produseres av disse sensorplattformene. I denne forbindelse har det vært veldig fruktbart å få erfaringer gjennom denne oppgaven med VR-briller som operatørgrensesnitt og utforske mulighetene dette gir.

Underveis har vi oppdaget at både oppløsning på kamera og briller kan være en begrensende faktor. Dette er det viktig å utforske videre, først og fremst med bedre kameraer. I denne sammenheng er det essensielt å ha et tett samarbeid med operativt personell, så man kan gjøre tilpasninger mot reelle operative krav. Videre tror vi det er viktig å prøve å utnytte automatisk prosessering i størst mulig grad, for å lette den kognitive byrden på operatøren. Dette er et felt vi forsker mye på både innenfor autonome systemer og situasjonsforståelse, og det er naturlig å bruke disse forskningsresultatene til denne typen prosjekter.

Rønningen og Kjernaas har levert et imponerende stykke arbeid med en fungerende demonstrator som viser konseptet fra ende til ende. Både VR-briller og 360-kameraer er foreløpig forholdsvis umodne produkter og verktøyene som følger med er på et tidlig stadium. Dette gjør det til en utfordring bare å få disse til å fungere hver for seg, så å få alle komponentene til å fungere sammen i sanntid er naturlignok vesentlig mer krevende. Her har det vært tekniske utfordringer hele veien, og det er helt klart at begge kadettene har vokst mye både som teknikere og utviklere. På tross av at de i utgangspunktet satte seg en hårete målsetning, er det liten tvil om at målene er oppnådd over forventning.

8 Veien videre

Vi som utvikler av VRSS ser for oss at fremtiden kan by på mange muligheter. Etter de begrensninger vi satte oss i begynnelsen har resultatet blitt over all forventning. Når det er sagt mener vi også at produkter har potensiale for enda flere nyttige funksjoner. Figur 2.4.1 i introduksjonen viser at vi innledningsvis i prosjektet hadde enda flere tanker og ideer enn vi rakk å ta for oss i første prototype.

VRSS er et produkt som allerede neste høst vil videreutvikles i en ny bacheloroppgave her ved Sjøkrigsskolen. Da vil kadett Trygve Tveiten ta steget å bringe produktet ut i en mer operativ kontekst. Det vil videreutvikles både innenfor hardware, og software. Et verktøy for distansemåling og målangivelse gjennom VR-brillene vil bli undersøkt, samtidig som monokkelfeste til hjelm vil bli konstruert.

Når det gjelder kamerabiten ønskes det i neste omgang å teste brillene med et skikkelig høyoppløsningskamera slik at grafikken på liveoverføringen inne i brillene blir vesentlig bedre. Et alternativ er å prøve prosjektet opp mot 360 kameraet og programvaren FFI har utviklet som oppfatter pikselendring i bilde. Dette gir en bedre måloppdagelse enn menneskeøyet. Et annet alternativ er at Kongsberg Protech kommer inn i prosjektet, og at VRSS forhåpentligvis kan testes sammen med deres Remote Weapon Station.

9 Bibliografi

9.1 Monografier

Kannala Juho, Brandt Sami S.

2006, ***A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses.*** Finland 19. Juni

Meen, Knut

Om Kalmanfilter, Sjøkrigsskolen, Bergen

9.2 Websider

Andresen, Geir

2010, Akselerometer. Det store norske leksikon, 18. September 2010

https://snl.no/akselerometer (24.11.2010)

Buckley, Sean

2015, *This is how Valves Amazing Lighthouse tracking Technology works*, Gizmodo. 19. Mai.

http://gizmodo.com/this-is-how-valve-s-amazing-lighthouse-tracking-technol-1705356768 (26.11.2016)

Digital Trends Staff

2016, *oculus-rift-vs-htc-vive.* Digital Trends, 16. Oktober

http://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive/ (25.11.2016)

HTC

2016, *VIVE READY computers*, www

https://www.vive.com/us/ready/(26.11.2016)

Johansen, Narve E,

2014, *Kartprojeksjoner*, Universitet i Oslo.

http://www.uio.no/studier/emner/matnat/math/MAT4010/v14/filer/mat4010---2-kartprojeksjoner.pdf (29.11.2016)

Kvalheim, Finn Jarle, Urke Eirik Helland

2016, Four 360 *cameras comared*, Teknisk Ukeblad Media, 18. august

http://www.tek.no/artikler/test-360-kamera-fra-samsung-lg-ricoh-og-giroptic/349860/2 (26.11.2016)

Opsahl, Thomas

2016, *The perspective camera*, Universitet i Oslo

http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html (28.11.2016)

Ormstad, Heimut

2009, *Gyroskop.* Det store norske leksikon, 14. Februar

https://snl.no/gyroskop (26.11.2016)

Rossen, Eirik

2009, *Rendering.* Det store norske leksikon, 14. September

https://snl.no/rendering%2FIT (28.11.2016)

Skytt, Lasse, Arvanaghi Babak

2016, *Virtual reality – fremtiden er her allerede.* Illustrert Vitenskap, 08. Juni

http://illvit.no/teknologi/gadgets/virtual-reality (20.11.2016)

VR Dev School

2016, *Vive developer Mini Course*, www

http://learn.vrdev.school/p/vive-developer-mini (26. Juli 2016)

Wikipedia Homepage

2016, *Fisheye lens*, www

https://en.wikipedia.org/wiki/Fisheye_lens (27.11.2016)

2016, *Miltiple camera setup*, www

https://en.wikipedia.org/wiki/Multiple-camera_setup (27.11.2016)

2016, *Fisheye Projection*, www

http://wiki.panotools.org/Fisheye_Projection (27.11.2016)

2016, *Unity (game engine)*, www

https://en.wikipedia.org/wiki/Unity_(game_engine) (27.11.2016)

2016, *Kalman filter,* www
https://en.wikipedia.org/wiki/Kalman_filter (26.11.2016)

9.3 Programvare

2016, HTC Vive, www

(https://www.htcvive.com/us/setup/)

2016, Unity, www

(https://store.unity.com/?_ga=1.240372042.814995130.1469539933)

2016, Steam www

(http://store.steampowered.com/about/)

2016, SteamVR, www

(https://support.steampowered.com/kb_article.php?ref=5254-FJKZ-7829)

2016, SteamVR plugin, www

(https://www.assetstore.unity3d.com/en/#!/content/32647

9.4 Figurer

Fisheyelinse:
http://thefuturephotographer.com/fisheye-lens-guide/

Linsebryting:
https://en.wikipedia.org/wiki/Lens_(optics)#/media/File:Lens3.svg

Perspektivkameramodell:
http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html

Intrinsikke del:
http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html

Matrisemultiplikasjon:
http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html

Euklidskvektor:
http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html

Komplett uttrykk Perspektivkameramodell:
http://maskinsyn.unik.no/Lecture_1_4_The_perspective_camera_model/Lecture_1_4_The_perspective_camera_model.html

Fisheye:
http://www.ee.oulu.fi/mvg/files/pdf/pdf_697.pdf

Theta – pi 3D:
https://en.wikipedia.org/wiki/Spherical_coordinate_system#/media/File:3D_Spherical.svg

Normaliserte bildekoordinater:
http://www.ee.oulu.fi/mvg/files/pdf/pdf_697.pdf

Light house tracking:
http://img.gfx.no/1863/1863886/HTC-Vive.jpg
www.tek.no

Kalmanlikning:
Hentet fra **Meen, Knut**
Om Kalmanfilter, Sjøkrigsskolen, Bergen

HTC VIVE:
http://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive/

Ricoh Theta S:
http://www.lydogbilde.no/test/foto-video/gruppetest_fire-360-kameraer/ricoh-theta-s

Unity
https://en.wikipedia.org/wiki/Unity_(game_engine)

webkamera:
https://s.aolcdn.com/hss/storage/midas/ca2687b9f4d4c49b314e73ab7dc18c2e/201745630/Full bleed+1.jpg

webkamera:
http://cdn.ndtv.com/tech/images/htc_vive_consumer_edition_front.jpg

Rådata 360-kamera:
http://www.gannett-cdn.com/-mm-/3c12e6a398b55d472a3a09f14ab8ac4987e435a2/r=880/http/videos.usatoday.net/Brightcove2/29906170001/2015/10/29906170001_4561944777001_video-still-for-video-4561903006001.jpg

Presentasjon 360 video:

http://360rumors.blogspot.no/2016/08/eyeforce-vr-headset-with-210-degree.html

Demonstrasjon 360 video:

http://paulbourke.net/dome/iDome/

Ekvirektangulært bilde:

http://paulbourke.net/miscellaneous/sphericalpano/

10 VEDLEGG

10.1 Vedlegg Manuell bildeinnhenting.

```
using UnityEngine;
using System.Collections;
using System.IO;
using UnityEngine.UI;


public class bilde : MonoBehaviour {
    public static Texture2D LoadPNG(string filePath)
    {
        if (File.Exists(filePath))
        {
            byte[] data = File.ReadAllBytes(filePath);

            Texture2D tex = new Texture2D(2, 2);
            tex.LoadImage(data);

            return tex;
        }

        return null;
    }


    // Use this for initialization
    void Start () {
        Renderer r = GetComponent<Renderer>();
        r.material.mainTexture = LoadPNG("C:\\bilde.jpg");
       }

       // Update is called once per frame
       void Update () {

       }
}
```

10.2 Vedlegg Automatisk bildeserie.

- Vi brukte følgende kode:

```
using UnityEngine;
using System.Collections;
using System.IO;
using UnityEngine.UI;


public class bilde : MonoBehaviour
{
    public Texture2D[] images = new Texture2D[4];
    private int counter = 0;
    private int currentImageINDEX = 0;

```

```
-
-       public static Texture2D LoadPNG(string filePath)
-       {
-
-           if (File.Exists(filePath))
-           {
-               byte[] data = File.ReadAllBytes(filePath);
-
-               Texture2D tex = new Texture2D(2, 2);
-               tex.LoadImage(data);
-
-               return tex;
-           }
-
-           return null;
-       }
-
-
-       // Use this for initialization
-       void Start()
-       {
-           images[0] = LoadPNG("C:\\bilde.jpg");
-           images[1] = LoadPNG("C:\\bilde2.jpg");
-           images[2] = LoadPNG("C:\\bilde3.jpg");
-           images[3] = LoadPNG("C:\\bilde4.jpg");
-
-           SetImage();
-       }
-
-       void SetImage()
-       {
-           Renderer r = GetComponent<Renderer>();
-           r.material.mainTexture = images[currentImageINDEX];
-       }
-
-       // Update is called once per frame
-       void Update()
-       {
-           ++counter;
-
-           if (counter >= 30)
-           {
-               counter = 0;
-
-               ++currentImageINDEX;
-
-               if (currentImageINDEX >= 4)
-               {
-                   currentImageINDEX = 0;
-               }
-
-               SetImage();
-           }
```

```
        }
    }
```

10.3 Vedlegg Rådata fra Webkamera.

```
using UnityEngine;
using System.Collections;
using System.IO;
using UnityEngine.UI;


public class bilde3 : MonoBehaviour
{
    public static Texture2D LoadPNG3(string filePath)
    {
        if (File.Exists(filePath))
        {
            byte[] data = File.ReadAllBytes(filePath);

            Texture2D tex = new Texture2D(2, 2);
            tex.LoadImage(data);

            return tex;
        }

        return null;
    }


    // Use this for initialization
    void Start()
    {
        WebCamTexture webcamTexture = new WebCamTexture();
        WebCamDevice[] devices = WebCamTexture.devices;

        webcamTexture.deviceName = devices[1].name;
        Renderer r = GetComponent<Renderer>();
        r.material.mainTexture = webcamTexture;
        webcamTexture.Play();
    }

    // Update is called once per frame
    void Update()
    {

    }
}

```

10.4 Vedlegg Skybox texture.

```
using UnityEngine;
using System.Collections;
using System.IO;
using UnityEngine.UI;


public class kamera : MonoBehaviour
{
    public static Texture2D LoadPNG4(string filePath)
    {
        if (File.Exists(filePath))
        {
            byte[] data = File.ReadAllBytes(filePath);

            Texture2D tex = new Texture2D(2, 2);
            tex.LoadImage(data);

            return tex;
        }

        return null;
    }


    // Use this for initialization
    void Start()
    {
        MovieTexture texture = Resources.Load<MovieTexture>("Demovideoogg");
        RenderSettings.skybox.mainTexture = texture;
        texture.Play();
    }


    // Update is called once per frame
    void Update()
    {

    }
}
```

10.5 Vedlegg Orientering virtuelt rom.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.VR;
using System.Collections;

public class Direction : MonoBehaviour {
    private SteamVR_Camera cam;
    private bool useDegrees = false;
    private Quaternion DefaultRotation=Quaternion.identity;


       // Use this for initialization
```

```
        void Start () {
         cam = SteamVR_Render.Top();
        }

        // Update is called once per frame
        void Update () {
         if (Input.GetKeyDown("j"))
         {
             useDegrees = true;
         }

         if (Input.GetKeyDown("k"))
         {
             useDegrees = false;
         }

         VRNode eye = VRNode.CenterEye;
         Vector3 pos = InputTracking.GetLocalPosition(eye);
         Quaternion rot = Quaternion.Inverse(DefaultRotation) *
InputTracking.GetLocalRotation(eye);
         float heading = rot.eulerAngles.y;

         int direction = Mathf.FloorToInt((heading / 360 * 12 + 11.5f) % 12) + 1;
         string directionText = "" + direction;
         if (useDegrees)
         {
             directionText = "" + Mathf.FloorToInt(heading) + "°";
         }


         GetComponent<TextMesh>().text = directionText;

         if (Input.GetKeyDown("n"))
         {
             DefaultRotation=InputTracking.GetLocalRotation(eye);
         }
    }
}
```

10.6 Vedlegg Kartorientering.

```
using UnityEngine;
using UnityEngine.VR;
using System.Collections;

public class MapHeading : MonoBehaviour
{
    private SteamVR_Camera cam;

    // Use this for initialization
    void Start()
    {
        cam = SteamVR_Render.Top();
    }

    // Update is called once per frame
    void Update()
    {
        VRNode eye = VRNode.CenterEye;
```

```
        Vector3 pos = InputTracking.GetLocalPosition(eye);
        Quaternion rot = InputTracking.GetLocalRotation(eye);
        GetComponent<Renderer>().gameObject.transform.localEulerAngles = new
Vector3(90, rot.eulerAngles.y, rot.eulerAngles.z);
        GetComponent<Renderer>().gameObject.transform.position = pos + (new Vector3(0,
-90, 0));
    }
}
```

10.7 Vedlegg Gjennomsiktighet.

```
using UnityEngine;
using System.Collections;

public class changealpha : MonoBehaviour {

    public KeyCode increaseAlpha;
    public KeyCode decreaseAlpha;
    public float alphaLevel = .5f;


       // Use this for initialization
       void Start () {

       }

       // Update is called once per frame
       void Update () {
        if (Input.GetKeyDown(increaseAlpha))
            alphaLevel += .05f;

        if (Input.GetKeyDown(decreaseAlpha))
            alphaLevel -= .05f;
        GetComponent<SpriteRenderer>().color = new Color(1, 1, 1, alphaLevel);
    }
}
```

10.8 Vedlegg Kjøretøysorientering.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.VR;
using System.Collections;

public class CarHeading : MonoBehaviour
{
    private SteamVR_Camera cam;
    private Quaternion DefaultRotation = Quaternion.identity;
    // Use this for initialization
    void Start()
    {
        cam = SteamVR_Render.Top();
    }

    // Update is called once per frame
    void Update()
    {
        VRNode eye = VRNode.CenterEye;
```

```
        Quaternion rot = Quaternion.Inverse(DefaultRotation) *
InputTracking.GetLocalRotation(eye);
        GetComponent<Renderer>().gameObject.transform.localEulerAngles = new Vector3(-
rot.eulerAngles.y, 90, 90);
        if (Input.GetKeyDown("n"))
        {
            DefaultRotation=InputTracking.GetLocalRotation(eye);
        }
    }

}
```

10.9 Vedlegg FFI søknad

Bacheloroppgave fra Sjøkrigsskolen: Virtual Reality(VR) styring av RWS/SEA PROTECTOR

Formalia

Ett makkerpar vil gjennomføre oppgaven sammen.
Navn: Jostein Kjernaas
Erlend Rønningen

Begge utdannes til våpenteknikkere (elektronikk og data) ved Sjøkrigsskolen i Bergen. Vi er inne i siste år som kadett, og bacheloroppgave vil bli en stor del av siste semesteret.

Bakgrunn for oppgaven

Da vi har det privilegium av å bli utdannet sammen med norske operatører fra diverse spesialstyrke avdelinger kom vi i diskusjon med de rundt fjernstyrte våpenplattformer. Det er systemer disse operatørene selv har erfaring med fra operativ oppdragsløsning. Blant alle fordelene de nevnte ved ubemannede eller fjernstyrte våpensystemer bet vi oss merke i **en** bakdel.
Konstruksjon av situasjonsbilde i en stridsone basert på bilder hentet fra en skjerm gjorde at helhetsoversikten ble dårligere enn om det hadde vært menneskeøyne som samlet samme informasjon. Denne observasjonen la grunnlag for problemstillingen i bacheloroppgaven vår:

"Hvordan kan vi, som tekniker, ved hjelp av teknologiske hjelpemidler, hjelpe våre medsoldater til å opprettholde og forbedre helhetsoversikten, uten at soldaten selv eksponeres."

Oppgaven

Vår idè og forslag til løsning på denne problemstillingen, er å konstruere ett kamerasystem som opereres gjennom VR-briller. (VR-briller skaper ett virtuelt bilde for brukeren, som gir en følelse av at du ser virkeligheten fremfor deg) Tanken er å plassere ett "360" kamera på våpenplattformen. Kameraet skal sende live bilder av omgivelsene ned til en sikkert plassert

operatør som får disse bildene opp på sine VR-briller. Bildet operatøren mottar vil dekke alle vinkler rundt våpenplattformen, og jeg ønsker å programmere VR-brillene slik at operatøren kan manøvrere seg rundt i bildet ved bruk av naturlige hodebevegelser. På denne måten vil soldaten i større grad få følelsen av at han selv sitter oppe på våpenplattformen.

Begrensing av Oppgave:
For å gjøre oppgavene oppnåelig innen tidsfristen ønsker jeg å fokusere på kommunikasjonen mellom kamera og VR-brillene. Jeg er klar over at i en operativ setting ville ett slikt kamera trenge en gyrostabilisert plattform for å unngå vibrasjoner, men det har jeg ikke kapasitet til å konstruere. Hvis en slik plattform i fremtiden utvikles(eller at kameraet kan festes direkte på det allerede gyrostabilisert ildgivnigssystemet) tror jeg det også kan være aktuelt å programmere siktemidler og andre nyttige hjelpemidler direkte inn på VR-brillene, slik at operatøren har alt i samme bilde. Dette har jeg ikke som primærfokus, men hvis muligheten byr seg og tiden strekker til er ett slikt siktemiddel også ønskelig å jobbe med.

- Frist for levering av oppgave: Desember 2016.

Hvorfor Kongsberg Protech Systems?

Grunnen til at jeg har kontaktet "Kongsberg Protech Systems" er i ønske om å opprette ett samarbeid rundt et slikt produkt, særlig rettet mot deres nåværende RWS. Jeg tror slik styring av deres våpenplattform kan gjøre systemet bedre og gi operatøren økt helhetsoversikten.

Jeg ønsker med dette å forhøre meg om dere i KPS kan være interessert i å bidra med støtte til min oppgave. Typisk form for bidrag kan være utlån av nødvendig utstyr, innkjøp av utstyr, og/eller tilgang til nødvendige ressurser/kompetanse. Gjennom Sjøkrigsskolen får hver enkelt kadett 20 000 kr disponibelt for oppgaven. Jeg tror dog at det alene kan bli for lite i en slik oppgave.

Som dere ser over har vi vært i kontakt med KPS på Kongsberg. Vi har drevet en god del planlegging av oppgaven allerede, men de viste seg denne forrige uke at Kongsberg ikke hadde kapasitet til å følge opp Bacheloroppgaver i høstsemesteret. Derfor retter vi oss mot dere i FFI, og hører om dere kanskje kunne tatt over ballen der Kongsberg slapp. Vi ser for oss akkurat samme oppgaven, og egentlig søker vi akkurat samme hjelpen. Det er snakk om økonomiske støtte til ett 360 kamera og VR-briller. Vi har 20 000 disponibelt, som nevnt over, men vi vil nok trenge ett økonomisk bidrag utenom dette. Hvorvidt Kongsberg som eier av RWS ikke er med i selve oppgaven ser vi ikke som noen hindring. Vi kan fortsatt lage systemet, å få dette til å fungere uavhengig om vi har en RWS tilgjengelig eller ikke. Tanken er at vår ide kan plasseres oppe på våpenplattformer, droner eller andre steder der du ønsker å simulere at du har 2 menneskeøyne tilstede.
Vi ser også for oss at flere VR-briller kan kobles inn i samme bilde, slik at du i spesielt krevende situasjoner kan bemannet fire soldater med VR-briller, som observerer og dekker hver sin sektor utenfor kjøretøyet eller fartøyet uten at de eksponeres.

Oppgaven skal gjennomføres av Jostein og Erlend, men vi har opprettet ett team her på skolen bestående av tidligere stridsbåt 90 sjefer, sjef kystjegerkommandoen(spesialstyrker) og operatører fra både Marinejegerkommandoen(MJK) og KJK. Deres rolle blir å bidra med praktisk erfaring slik at brukersnittet står i samsvar med det de mener trengs ute i felten. Operatørene stiller også opp for testing, særlig når det gjelder sjøsyke, eller brukervennlighet i stressende situasjoner.

Vi har allerede i samsvar med teamet på skolen kommet frem til en rekke nyttige sensorer som kan programmeres inn i systemet for å gjøre jobben enklere for soldatene ute i strid. Spesialstyrkene ser ett stort behov for ett sånt system, særlig i forbindelse med RWS, og potensialet mener vi er stort for at en slik plattform kan videreutvikles til en virkelig suksess. Vi tror at VR-briller kommer til å komme inn i forsvaret i nær fremtid uavhengig av dagens situasjon.
Vi er klar over problemene knyttet til sjøsyke og kvalme, men i samarbeid med spesialstyrkene håper vi at vi dette problemet kan løses, eller trenes opp til å handteres.

Håper på positiv respons!

Med vennlig hilsen
Fenrik Jostein Kjernaas
Sjøkrigsskolen Bergen