Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i Datateknologi	Vårsemesteret, 2021 Åpen / Konfidensiell
Forfattere: Brage Solheim, Erik Finnesand og Sander S. Jørgensen	Signatur forfattere: Brage Solheim Erik Finnesand Sander S. Jørgensen
Fagansvarlig: Erlend Tøssebro Veileder(e): Erlend Tøssebro	
Tittel på bacheloroppgaven: Projisering av geologiske modeller i en sandkasse, et brukergrensesnitt utviklet for GemPy og Open AR Sandbox Engelsk tittel: Projecting geology onto a sandbox, a friendly and functional user interface for GemPy and the AR sandbox	
Studiepoeng: 20	
Emneord:	Sidetall: 77 + vedlegg/annet: 3 Stavanger, 31/05/2021 dato/år

Projisering av geologiske modeller i en sandkasse, et brukergrensesnitt utviklet for GemPy og Open AR Sandbox

Bachelor i Datateknologi

Erik Finnesand
Sander Sandanger Jørgensen
Brage Solheim

Veiledet av
Erlend Tøssebro

Universitetet
i Stavanger

Institutt for data- og elektroteknologi
Det teknisk-naturvitskaplege fakultet
Mai 2021

Sammendrag

Gruppen fikk tildelt en oppgave hvor formålet var å videreutvikle programvaren Open AR Sandbox[1]. Et simuleringsprogram som bruker en sensor og projektor til å vise ulike moduler i en fysisk sandkasse. Idéen er at en bruker former et landskap i sandkassen, sensoren leser av landskapet, og en datamaskin bruker projektoren til å projisere bildet ned i sandkassen. Målet var å lage et brukergrensesnitt til programvaren, samt å bygge ut nye moduler som var relevante for geologistudentene ved Instituttet for Energiressurser ved Universitetet i Stavanger.

Oppgaven ble utlyst av Universitetet i Stavanger. Gruppen sto fritt til å velge hvordan oppgaven skulle løses og hva som skulle legges til i programmet. Oppgaven ble løst i dialog med oppgaveansvarlig, professor Nestor Fernando Cardozo Diaz. Oppgaveansvarlig tildelte gruppen en liste med funksjonaliteter som var ønsket av instituttet. Listen fra oppgaveansvarlig inkluderte:

- Utvikle et nytt brukergrensesnitt til programvaren Open AR Sandbox, som gjør det enklere å bruke for studenter uten forkunnskaper i Python.
- Projisere geologiske modeller lagd med Python-biblioteket GemPy i sandkassen. Ideen er at sandkassen skal vise et geologisk kart basert på den geologiske modellen og landskapet formet i sanden.
- Utvikle et brukergrensesnitt som gjør det mulig å konstruere, modifisere og lagre 3D geologiske modeller i Gempy, uten forkunnskaper i Python.

Oppgaven ble løst ved å først analysere koden og funksjonene til den eksisterende programvaren. Etter dette startet gruppen på listen fra oppgaveansvarlig. De to hovedpunktene i listen var å utvikle et brukergrensesnitt til programmet, samt å utvikle en modelleringsmodul med et eget brukergrensesnitt. Modelleringsmodulen skulle brukes til å lage geologiske modeller og projisere modellene i sandkassen. Brukergrensesnittet til Open AR Sandbox er tilegnet å være en brukervennlig platform for geologistudenter uten forkunnskaper i Python.

Gruppen har laget programmet slik at det skal være enkelt å videreutvikle ved et senere tidspunkt.

Forord

Vi vil gi en stor takk til vår veileder førsteamanuensis Erlend Tøssebro for god veiledning gjennom prosjektet. Vi vil også takke professor Nestor Fernando Cardozo Diaz for dialogen rundt utviklingen av GemPy modelleringsmodulen, skript for geologiske utregninger og en solid innføring i geologi. Dette kommer vi til å ta med oss på veien videre. Vi vil gi en stor takk til sjefsingeniør ved Universitet i Stavanger Ståle Freyer for bistand i konstruksjon av aluminiumsprofilen som ble brukt til å montere dybdesensoren. Vi vil gjerne takke Paul Hjelm for filming og redigering av demo-filmen til programmet.

Til slutt vil vi takke Daniel Escallón Botero fra Open AR Sandbox-teamet for hjelp med å feilsøke problemer som oppsto ved oppsett av Kinect v1 og hjelp med å konfigurere ArUco-merker.

Nomenklaturliste

AR - Utvidet virkelighet
CSS - Cascading Style Sheets
DEM - Digital Elevation Model
URL - Uniform Resource Locator
JSON - JavaScript Object Notation
CSV - Comma-separated values
IR - Infrarød
UiS - Universitetet i Stavanger
USB - Universal Serial Bus
2D - Todimensjonal
3D - Tredimensjonal
IDE - Integrert utviklingsmiljø
BAT - Satsvis fil
HTTP - Hypertext Transfer Protocol
API - Programmeringsgrensesnitt

Innhold

1 Innledning

1.1 Oppgavebeskrivelse

Oppgaven består av å videreutvikle programvaren Open AR Sandbox. Open AR Sandbox er en utvidet virkelighets-programvare som bruker en dybdesensor og en projektor til å plotte og visualisere geologiske data i en sandkasse. Programvaren er utviklet av forskningsenheten "Computational Geoscience and Reservoir Engineering" ved RWTH Aachen University i Tyskland, og er ment til bruk i geofaglig utdanning. Programmet består av 11 moduler, samt moduler som brukes til å manuelt kalibrere dybdesensor og projektor[1].

Det skal utvikles et brukergrensesnitt og en modelleringsmodul som skal brukes av geologistudenter til å lese og visualisere data i sanntid. Modelleringsmodulen skal ha egenskapen til å kunne modellere forskjellige geologiske modeller og projisere disse modellene i sandkassen.

1.2 Mål

Målet for oppgaven kan bli delt inn i tre deler:

- Utvikle et brukergrensesnitt til Open AR Sandbox som er enkelt å bruke for personer uten forkunnskaper i verken Python eller Jupyter Notebook. Videre, ønskes det at brukergrensesnittet er forståelig for både norske og internasjonale studenter.
- Utvikle et modelleringsverktøy for geologiske modeller. Disse geologiske modellene skal kunne projiseres i sandkassen. Modelleringsverktøyet skal bruke et lettfattelig brukergrensesnitt og være tilgjengelig gjennom brukergrensesnittet til Open AR Sandbox.
- Importere og eksportere modeller til og fra andre programmer.

1.3 Motivasjon

AR-programvare har de siste årene blitt mer og mer brukt blant høyskoler i hele verden. Ifølge en rapport fra "Hindawi" som testet AR-programvare blant studenter, økte oppmerksomheten blant studentene med 30,72%. Tilliten blant studentene økte med 10,74% og tilfredsheten økte med 12,50%[2].

Det er ikke lett å visualisere geologisk data fra et todimensjonalt bilde. Motivasjonen for oppgaven var å gi geologistudentene ved UiS et interaktivt og digitalt

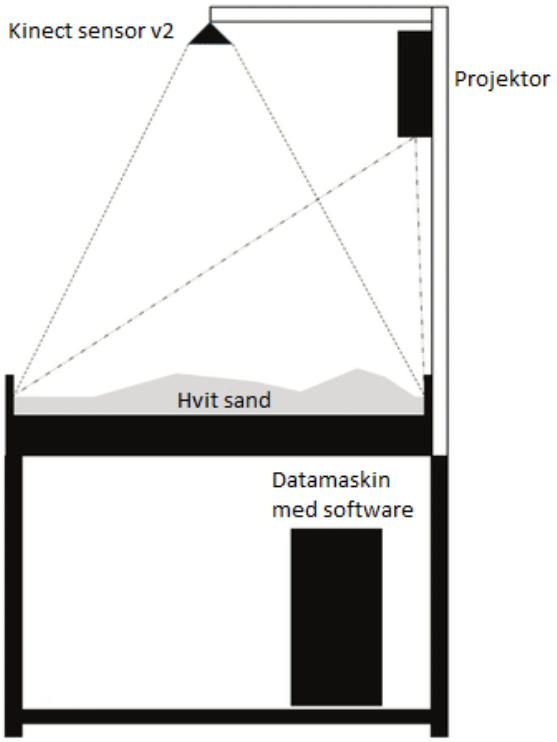
læringsverktøy slik at de lettere kan visualisere geologiske hendelser og geologiske data i 3D.

Open AR Sandbox må startes ved bruk av terminalvinduet, dette kan oppleves som tungvint for uerfarne brukere. Det finnes heller ikke noe eksisterende brukergrensesnitt for GemPy (Python-biblioteket som blir brukt for modellering). Siden begge programvarene bruker åpen kildekode er en annen motivasjon å gjøre programvaren tilgjengelig for alle som bruker Open AR Sandbox eller ønsker å lage geologiske modeller i GemPy. Prosjektet publiseres derfor offentlig på GitHub.

2 Open AR Sandbox

Dette kapittelet tar for seg sandkassens fysiske oppsett og funksjonaliteten til Open AR Sandbox. Det vil også bli forklart hvilke moduler som er tilgjengelig, samt deres funksjonaliteter.

2.1 Fysisk oppsett



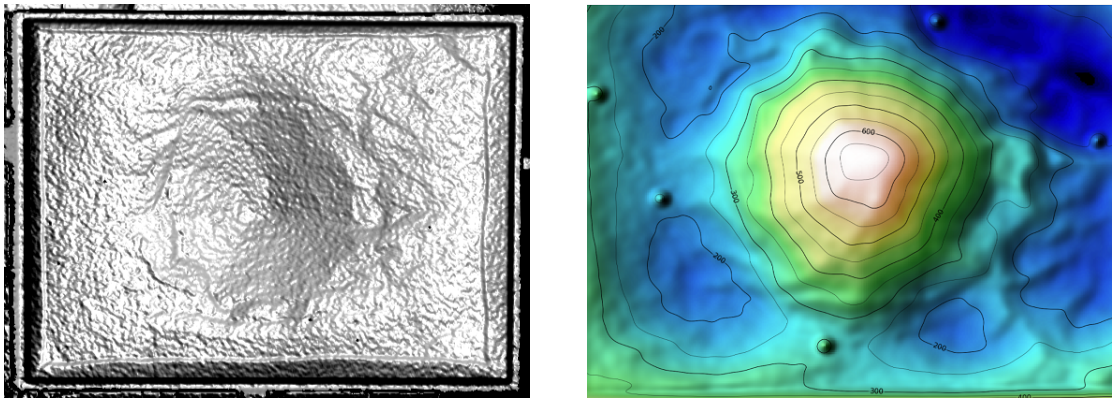
Figur 1: Fysisk oppsett av sandkassen[3]

Utstyret som trengs for å ta i bruk Open AR Sandbox er illustrert i figur 1 ovenfor. Sensoren registrerer avstanden til sanden for å danne et dybdebilde av overflaten. For at sensoren skal fange opp mest mulig reflekterende lys brukes det hvit sand. Den hvite sanden gjør det også lettere å se de ulike fargekonturene som projektoren projiserer ned i sandkassen. Dimensjonen på sandkassen er 100x80cm. Sensoren og projektoren er sårbare for bevegelse, derfor er det viktig at stativet holdes i ro under bruk. Projektoren og sensoren er festet til sandkassen ved hjelp av en aluminiums-profil. For å feste sensoren til aluminiumsprofilen ble det brukt en dreiet stålprofil. Stålprofilen er konstruert slik at sensoren enkelt kan fjernes i fremtiden om det skulle være behov for endring av oppsettet.

2.2 Programvare

I dette delkapittelet forklares programvaren Open AR Sandbox som ble utviklet av forskningsenheten ved RWTH Aachen University. Kapitellet dekker funksjonalitet, brukergrensesnitt, serveren som brukes, samt hva de ulike modulene gjør.

2.2.1 Funksjonalitet



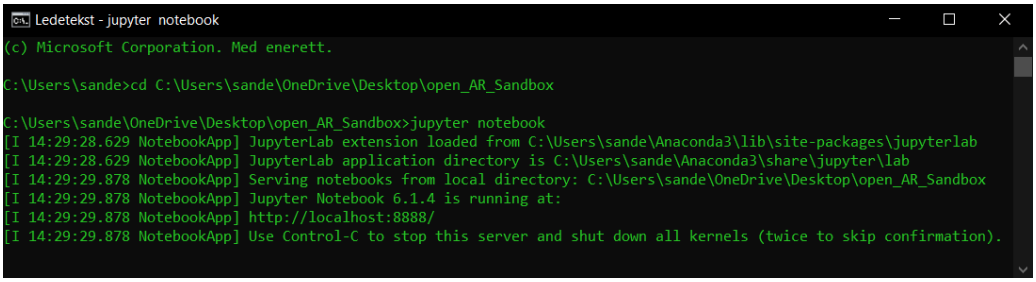
(a) Sensorbilde av sandkassen (b) Projektorbilde av sandkassen

Figur 2: Sensorbilde og projektorbilde av samme topografi

Når Open AR Sandbox starter opp, laster programmet inn sensor- og projektor-funksjonene. Når disse funksjonene er lastet inn venter programmet på funksjonen som starter serveren. Funksjonen til serveren tar inn modulene som skal være aktive som argumenter, før den starter hovedtråden. Når hovedtråden er aktiv registrerer sensoren dybdebildet i sandkassen og sender denne dataen i form av et NumPy-array til de aktive modulene. Et NumPy-array er en matrise med verdier av samme type og indekseres av en tuppel med ikke-negative heltall[4]. Modulenes klasser beregner dataene de får av sensoren og sender de videre til projektor-klassen for å kunne projisere et geologisk bilde i sandkassen. Det geologiske bildet som projiseres i sandkassen oppdateres med endringer i sanden. Open AR Sandbox bruker som standard 0,01 millisekund på å plotte et objekt (en rett linje av en konturlinje, en høydemåling, retningsvektor osv.). Dette fører til at projektoren normalt vil ligge et par bilder bak dybdesensoren, hvilket gir en forsinkelse på alt fra noen millisekunder til rundt ett sekund avhengig av hvor mye data som projiseres i sandkassen. Plottingintervallet kan endres om det er ønskelig, men dette går på bekostning av prosessorbruk.

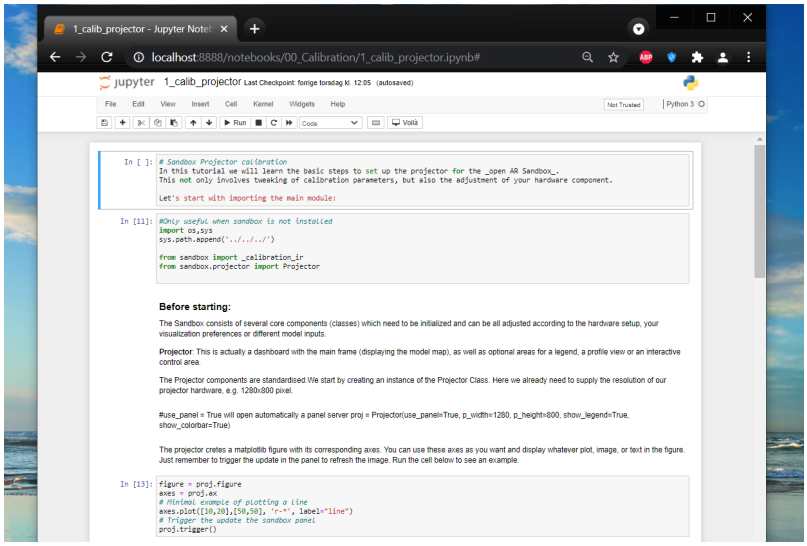
2.2.2 Jupyter Notebook

Open AR Sandbox bruker Jupyter Notebook som brukergrensesnitt. Jupyter Notebook er en nettapplikasjon som lar deg åpne, redigere og kjøre kode fra en nettleser[5]. Gjennom Jupyter Notebook kan brukere kjøre forskjellige moduler, kalibrere sensor og projektor, endre på innstillinger og starte serveren. Open AR Sandbox startes ved bruk at et terminalvindu. I terminalvinduet navigeres det til mappen med Notebook-filene, før serveren startes med kommandoen "jupyter notebook", som vist i figur 3.



Figur 3: Oppstart av Jupyter Notebook-server

Jupyter Notebook-serveren åpnes i et nettleservindu. Fra nettleservinduet navigeres det videre til modulen som skal brukes. Resultatet er som vist i figur 4.

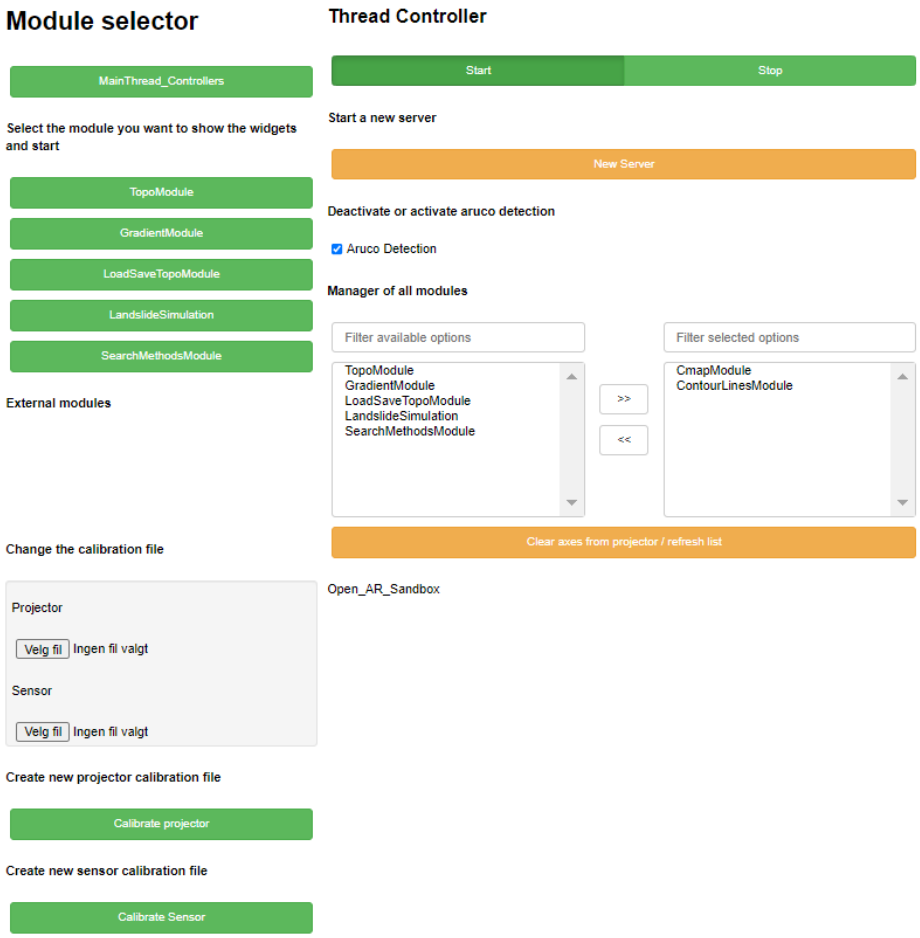


Figur 4: Kalibreringsmodulen åpnet i et nettleservindu

Denne oppstartsprosessen kan være både utfordrende og tidkrevende for geologistudenter uten noen erfaring med kode-utførelse. Av denne grunn var det behov for et nytt og utbedret brukergrensesnitt.

2.2.3 Server

Open AR Sandbox bruker en API for å kjøre serveren. En API er et grensesnitt som lar to ulike programvarer kommunisere. API-en bruker Python-biblioteket "Panel" for å kommunisere med serveren. Panel brukes til å lage nettapplikasjoner med støtte for interaktive widgets [6]. Serveren bruker Python-biblioteket "Bokeh". Bokeh brukes til interaktiv visualisering i nettlesere, som for eksempel konstruering av dashbord[7].



Figur 5: Kontrollpanel til server som kjøres i localhost

Når serveren starter åpner den to forskjellige porter på localhost. Disse to localhost-adressene blir åpnet i to respektive nettleservindu. Det ene vinduet inneholder kontrollpanelet til serveren, som vist i figur 5. Det andre vinduet innenholder bildet som projiseres ned i sandkassen, som vist i figur 2b. En localhost er den lokale nettverksadressen til datamaskinen.

2.2.4 Moduler

Open AR Sandbox består av flere moduler som kan kjøres hver for seg i egne Jupyter Notebook-vinduer eller samlet i kontrollpanelet.

Kalibrering av projektor og sensor
Før noen andre moduler kan initialiseres må projektoren og sensoren kalibreres. Dette blir gjort i to forskjellige Jupyter Notebook-filer hvor kalibreringsdataen blir lagret som JSON-filer. Ved kalibrering av projektor blir bredde, lengde og marginene til sandkassen justert for at projektorbildet skal dekke hele sandkassen. Ved kalibrering av dybdesensor gjøres det samme som ved kalibrering av projektor, men her kalibreres også maksimum- og minimum-høyde for dybdelesning av sandkassen.

MainThreadModule
MainThread-modulen er den overordnede modulen for alle modulene i programvaren og kjøres samtidig som en annen modul brukes. I modulen kan du endre fargekontrasten på bildet som skal projiseres i sanden og velge en lyskilde. Lyskilden skal etterligne sollys og kan brukes for å simulere skygger i topografien. Med topografi menes terrengforholdet i sandkassen. Lyskilden kan konfigureres ved bruk av en geolokasjon eller vinkel og høyde fra midtpunktet i sandkassen. Ved bruk at geolokasjoner kan man enten spesifisere et geografisk koordinat eller en by som for eksempel Stavanger. Lyskilden vil da bruke solens posisjon i det valgte området. Modulen gjør det også mulig å definere ønskelige konturintervaller som kan vises i sandkassen.

MarkerDetection
MarkerDetection-modulen bruker binære markører kalt ArUco-merker til å oppdage posisjon og orientering i sandkassen. ArUco-merkene som vises i figur 6 kan printes ut og plasseres i sanden for å utløse modelleringsaspekter som for eksempel:

- Krysseksjoner mellom to ArUco-merker
- Virtuelle borehull på en posisjon i sandkassen
- Utløse jordskredsimulasjoner
- Lokasjon av elektroder i geoelektriske simulasjoner

Figur 6: ArUco-merker[8]

AruCo-merkene plasseres i sanden og oppdages av dybdesensoren. Denne informasjonen sendes tilbake til programmet og brukes til ønsket formål.

ArUco-merker finnes i forskjellige bit-størrelser. For at programmet skal gjenkjenne ArUco-merkene er det viktig at de er av størrelsen 16 bits. De ulike bit-størrelsene ligger i egne oppslagslister. Open AR Sandbox bruker oppslagslisten "aruco.DICT_4X4_50". Oppslagslisten indikerer at størrelsen er 4x4 bit store, samt at det kan generes 50 varianter av disse ArUco-merkene. Hver variant har et ID-nummer fra 0-49.[9]. Hvordan dette ser ut i sandkassen er vist i figur 22.

Det er mulig å endre standard oppslagsliste hvis dette skulle være ønskelig. Dette må gjøres manuelt i filen aruco.py som ligger i Open AR Sandbox-mappen.

SearchMethodsModule
SearchMethod-modulen bruker dybdebildet til sandkassen for å utføre en Monte Carlo simulasjonsalgoritme. Simulasjonsalgoritmen brukes for å konstruere en sannsynlighetsfordeling basert på høyden av topografien i sandkassen. En Monte Carlo-simulasjon gjør mange tilfeldige prøvetakinger av et datasett for å løse umulige integraler[10].

GemPyModule
GemPy-modulen bruker Python-biblioteket GemPy for å laste inn og visualisere GemPy-modeller i sandkassen. GemPy gir en bruker mulighet til å konstruere en geologisk modell med flere tilpassede geologiske lag. Etter at en modell er lagret kan den lastes inn i GemPy-modulen og projiseres i sandkassen.

GradientModule
Gradient-modulen bruker gradientinformasjonen fra dybdebildet for å legge til skygger, visualisere vektorfelt eller en strømlinjegrafer i modellen. Vektorfeltetene starter på de høyeste punktene i topografien og går mot de laveste punktene i topografien.

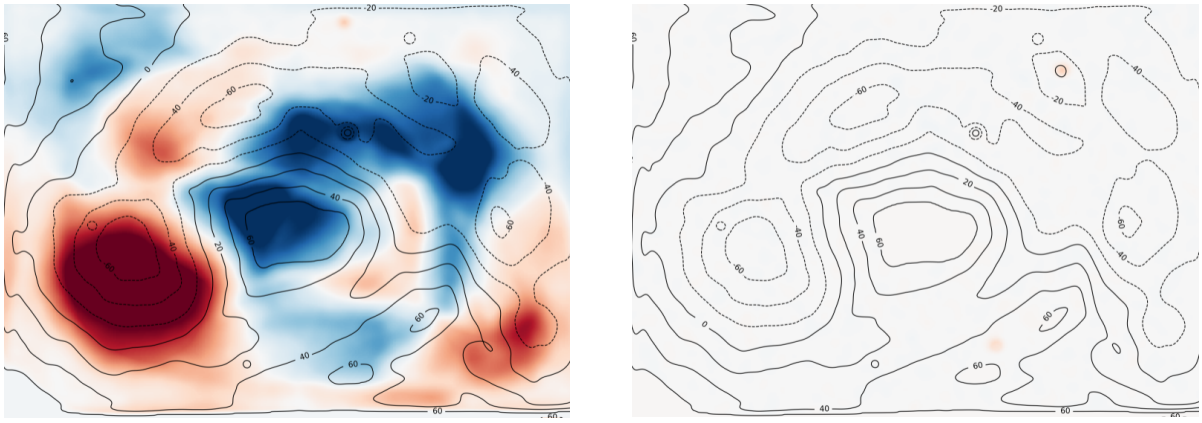
LoadSaveTopoModule
Denne modulen brukes til to formål:

1. Laste inn en topografisk modell i form av en DEM-fil i sandkassen, for å kunne gjenkonstruere modellen.

2. Lagre et dybdebilde av topografien som er konstruert i form av en DEM-fil, for å kunne bruke det ved en senere anledning.

DEM-filer er filer som inneholder høydeverdier i form av vektorpunkter eller piksler. DEM-filer brukes blant annet til 3D-modellering av topografier[11]. Topografimodellen kan være fra virkeligheten eller et dybdebilde av topografier man har konstruert tidligere. Måten dette gjøres på er at man laster inn topografimodellen i form av en DEM-fil. Projektoren vil så lyse opp hvilke sandområder som må endres for å gjenkonstruere topografien i sandkassen. Sandområdene lyses opp med farger, som vist i figur 7b.

- Rødt område indikerer hvor man skal fjerne sand.
- Blått område indikerer hvor man skal legg til sand.
- Hvitt område indikerer hvor topografien er korrekt.

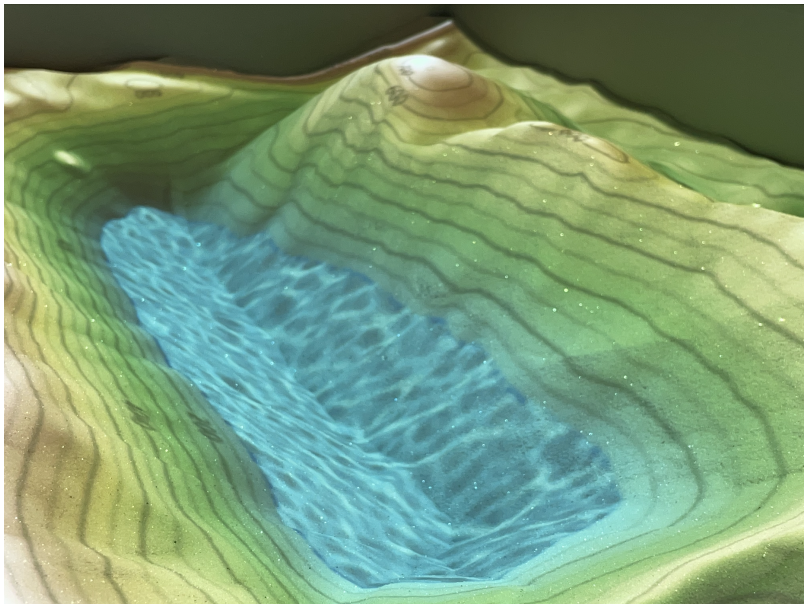


(a) Områder som må endres (b) Etter det har blitt lagt til sand

Figur 7: Bruk av LoadSaveTopoModule

Ved gjenkonstruering av en modell kan konturlinjene til modellen man skal bygge projiseres i sandkassen. Dette vil gjøre det lettere å gjenskape topografien.

TopoModule
Topografimodulen normaliserer dybdebildet og bruker dette til å lage et topografisk kart i sandkassen. Et topografisk kart beskriver et landområdes terrengforhold. Kartet innenholder konturlinjer og høydemålinger av forskjellige punkter i sandkassen. Modulen kan også simulere vann og fluiddynamikk i topografien, som vist i figur 8.



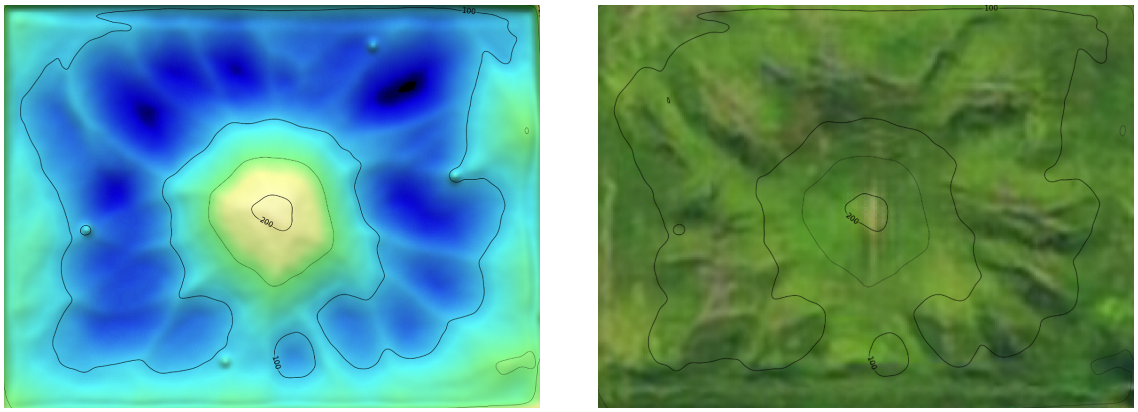
Figur 8: Topografi-modulen med vannivå

LandslideSimulation
Landslide-modulen gjør det mulig å simulere jordskred i sandkassen. Simulasjonen visualiserer flyt, retning og hastighet, enten i sanntid eller steg for steg. Open AR Sandbox kommer med fem forhåndsdefinerte topografiske modeller med flere forhåndsdefinerte jordskred. For å laste inn en av disse modellene i sandkassen brukes LoadSaveTopo-modulen

Ved bruk av ArUco-merker er det mulig å konstruere egne jordskred-simulasjoner. ArUco-merket definerer hvilket punkt i sandkassen jordskredet skal starte.

PrototypingModule
PrototypingModule er en Jupyter Notebook-mal som gjør det enklere å importere nye moduler til sandkassen. Malen består av de essensielle elementene alle moduler må ha med, som kalibrerings- og sensor-filer, samt MainThread-funksjonen.

LandscapeModule
Landscape-modulen bruker CycleGAN og pix2pix i Python-biblioteket PyTorch. Disse bibliotekene bruker maskinlæring til å generere landskap i sandkassen. Modulen bruker dybdesensoren til å registrere overflaten til sandkassen. Informasjonen generes om til en DEM-fil. DEM-filen blir modifisert av CycleGan og pix2pix, som generer et landskapsbilde. Dette bildet blir så projisert i sandkassen i sanntid. Per dags dato er det implementert 15 forskjellige landskapsbilder som kan brukes. I figur 9b har et satellittbilde blitt generert og lagt over den eksisterende topografien i sandkassen.

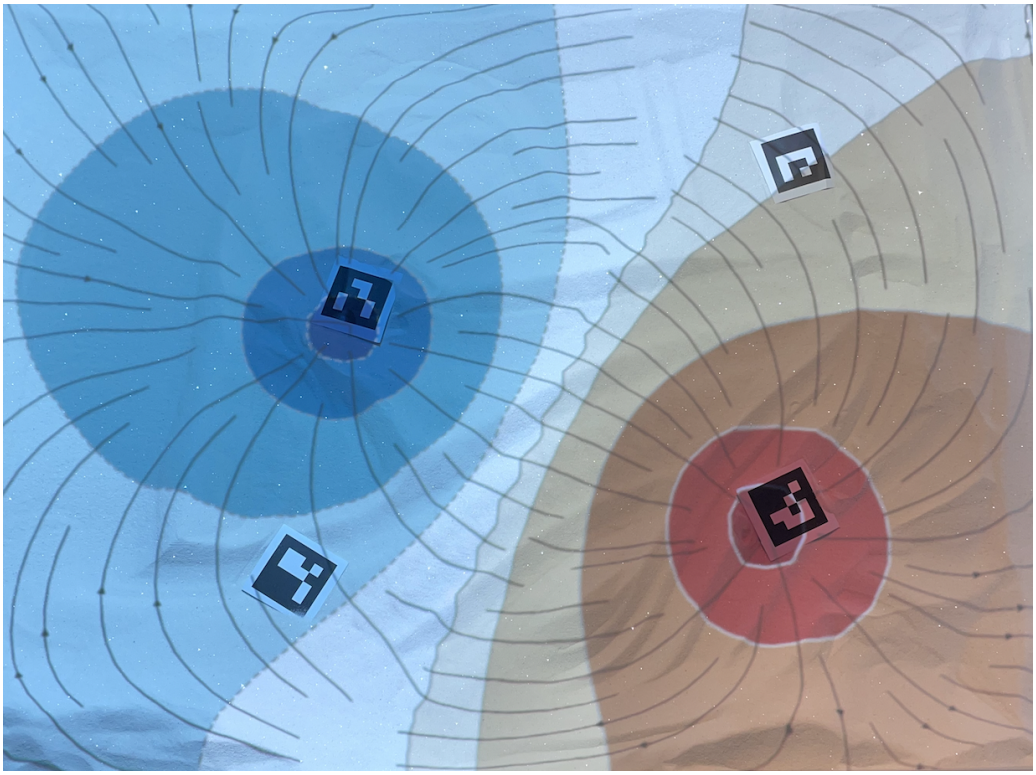


(a) Uten LandscapeModule (b) Med LandscapeModule

Figur 9: Samme topografi i sanden

SeismicModule
Bruker Python-biblioteket Devito for seismisk modellering i sandkassen. Denne modulen er kun tilgjengelig i Linux-baserte operativsystemer og ble derfor ikke brukt i oppgaven.

GeoelectricsModule
Geoelectrics-modulen bruker Python-biblioteket PyGIMLI til å visualisere elektriske felt i sandkassen. PyGIMLI brukes for modellering og inversjon innen geofysikk[12]. Det elektriske feltet defineres i sandkassen ved å plassere fire ArUco-merker, som etterligner elektrodene. ArUco-merkene blir tildelt en negativ eller positiv ladning ut ifra ArUco-merkenes ID. Elektrodeinformasjonen fanges opp av sensoren og sendes tilbake til modulen. Dette projiseres ned i sandkassen som et elektrisk felt. Det elektriske feltet viser negative poler i rødt og positive poler i blått. Hvordan dette ser ut i sandkassen er vist på bildet i figur 10.



Figur 10: Elektrisk felt projisert i sandkassen

På bildet i figur 10 er det fire ArUco-merker. Disse er for positiv ladning, negativ ladning, potensiell positiv ladning og potensiell negativ ladning. Differanse mellom de to potensielle ladningene definerer den elektriske spenningen i feltet[13].

3 Teknologi og Bibliotek

Dette kapittelet tar for seg teknologi og biblioteker som er brukt i oppgaven, samt hvor disse verktøyene brukes.

3.1 Python

Python er et høynivå objektorientert programmeringspråk[14]. Python ble brukt for å utvikle brukergrensesnittet til programmet og for å programmere GemPy modelleringsmodulen. Dette programmeringspråket ble valgt siden Open AR Sandbox allerede er skrevet i Python og siden Python har et stort utvalg av biblioteker og pakker.

3.1.1 Anaconda

I denne oppgaven ble det valgt å kjøre Python-distribusjonen Anaconda, ettersom utviklerne av Open AR Sandbox anbefalte dette. Anaconda er et distribusjonsprogram med over 25 millioner brukere på verdensbasis. Det er tilgjengelig over 7500 datavitenskaps-pakker og maskinslæringspakker som kan lastes ned gjennom Anacondas installasjonskommando "conda install[navn på ønsket pakke]". Ved bruk av Anaconda er det enkelt å lage flere forskjellige virtuelle miljø som kan kjøres separat[15].

Fordelen med å kjøre Open AR Sandbox gjennom et eget virtuelt miljø, er at programmet ikke blir påvirket av pakker i andre miljøer på datamaskinen. Anaconda inneholder også en rekke forhåndsinstallerte pakker og biblioteker, som ble benyttet i utviklingen av GemPy-modelleringsmodulen. Eksempler er NumPy, SciPy, Matplotlib og Jupyter Notebook[16].

3.1.2 GemPy

GemPy er et åpen kildekode Python-bibliotek som brukes til 2D- og 3D-geologisk modellering[17]. GemPy ble valgt som modelleringsbibliotek siden det ble oppgitt i oppgaven at GemPy skulle brukes. Ettersom GemPy er utviklet av den samme gruppen som har utviklet Open AR Sandbox, finnes det allerede en modul for projisering av GemPy modeller i programmet.

GemPy blir brukt i GemPy modelleringsmodulen for å lage modeller og projisere disse modellene til sandkassen.

3.1.3 PyQt

PyQt er et åpen kildekode Python-bibliotek som brukes til å utvikle brukergrensesnitt. PyQt er skrevet i C++, og er utviklet av det britiske firmaet Riverbank Computing[18]. Ved å bruke PyQt i kombinasjon med Python kan man utvikle applikasjoner uten å ofre for mye datakraft. PyQt består av over tusen klasser, noe som gir biblioteket stor fleksibilitet. PyQt støtter blant annet nettverkssockets, SQL databaser, XML, OpenGL og mange forskjellige typer widgets[19]. PyQt ble valgt på grunn av sitt mangfold av klasser og fleksibiliteten dette gir. I tillegg støtter PyQt CSS, som definerer utseendet til selve brukergrensesnittet.

PyQt ble brukt for utvikling av brukergrensesnittet til Open AR Sandbox og GemPy-modelleringsmodulen. Brukergrensesnittene ble laget i to forskjellige klasser og kan derfor fungere som enkeltstående applikasjoner. GemPy-modelleringsmodulen er også tilgjengelig i det nye brukergrensesnittet til Open AR Sandbox.

3.1.4 NumPy

NumPy er et åpen kildekode Python-bibliotek som blir brukt til å håndtere matematiske funksjoner som for eksempel matriseutregninger[20]. NumPy ble valgt siden det har støtte for matriseutregning.

NumPy blir brukt til tre funksjoner i GemPy-modelleringsmodulen. Den første funksjonen blir brukt til å rotere punkter i et koordinatsystem ved hjelp av en rotasjonsmatrise. Den andre funksjonen blir brukt for å finne skjæringspunktet mellom en linje og en plan. Den siste funksjonen brukes til å korrigere punkter som havner utenfor grensen til koordinataksen ved rotasjon av punkter i et koordinatsystem.

3.1.5 SciPy

SciPy er et åpen kildekode Python-bibliotek som blir brukt til vitenskapelig og teknisk databehandling[21]. SciPy ble valgt på grunn av at det har støtte for k-d trær. Et k-dimensjonalt tre er en romdelende datastruktur som blir brukt for organisering av punkter i et k-dimensjonalt rom[22]. Gitt en liste med punkter i et koordinatsystem, kan dette brukes til å finne hvilket av disse som er nærmest et annet spesifikt punkt.

SciPy ble brukt i en funksjon for å finne nærmest punkt(se kode 14). Funksjonen blir brukt i GemPy-modelleringsmodulen.

3.1.6 PyVista

PyVista er et åpen kildekode Python-bibliotek som ble brukt til visualisering av modeller i et 3D koordinatsystem[23]. PyVista ble valgt siden det er en kravspesifikasjon til GemPy.

PyVista blir brukt for å vise geologiske modeller i 3D i GemPy-modelleringsmodulen.

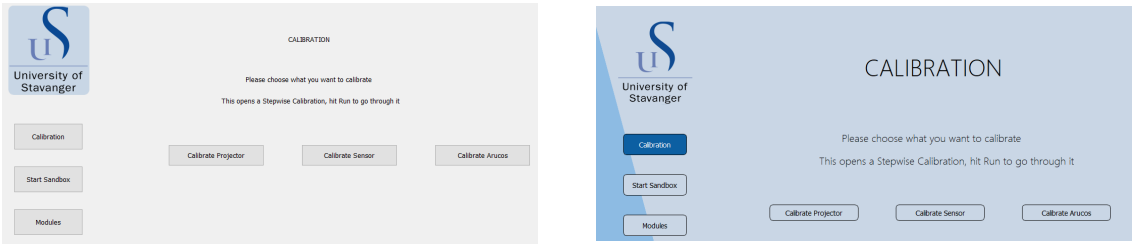
3.1.7 Matplotlib

Matplotlib er et åpen kildekode Python-bibliotek som blir brukt lage og vise modeller i et 2D koordinatsystem[24]. Matplotlib ble valgt siden det er en kravspesifikasjon til GemPy.

Matplotlib blir brukt for å vise geologiske modeller i 2D i GemPy modelleringsmodulen.

3.2 CSS

Cascading Style Sheets (CSS) er et gjennomgående stilark som brukes til å definere hvordan elementer skal gjengis på skjermen. CSS er et fleksibelt språk som tillater bruk av samme stil til å definere flere elementer[25]. Eksempelvis er stilen til knappene i applikasjonen gjennomgående lik. Grunnen til at det ble valgt å bruke CSS var fordi det støttes av PyQt og at brukere av applikasjonen skal kunne navigere seg rundt i applikasjonen på en enkel og oversiktlig måte.



(a) Før bruk av CSS

(b) Etter bruk av CSS

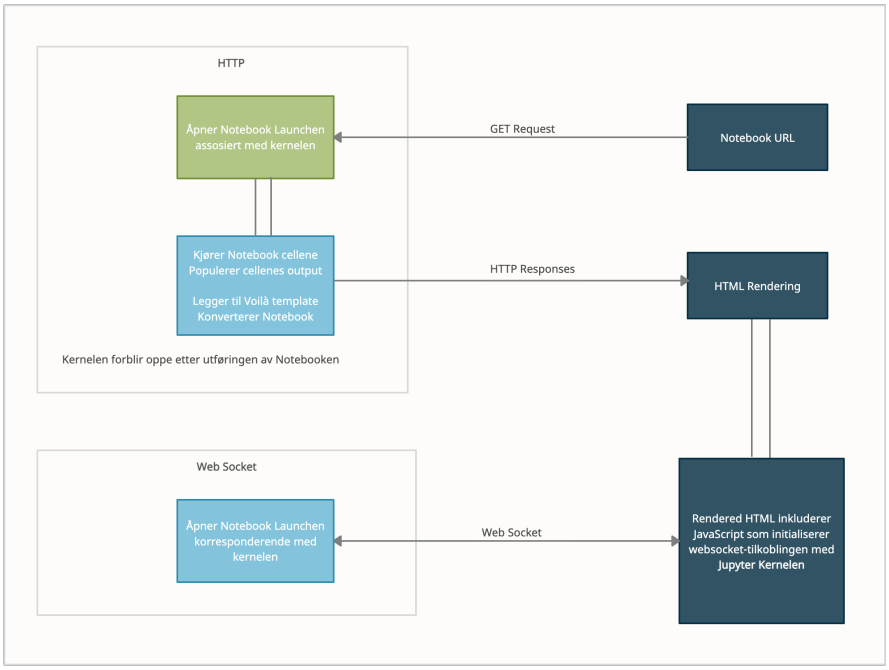
Figur 11: Manipulering av brukergrensesnitt ved bruk av CSS

3.3 Jupyter Notebook

Jupyter Notebook er en nettapplikasjon som brukes til å åpne redigere og kjøre kode fra en nettleser. Koden kjøres stegvis fra forskjellige celler i Jupyter Notebook-filen. Med Jupyter Notebook er det mulig å kjøre kode uten et integrert utviklingsmiljø. Open AR Sandbox bruker Jupyter Notebook til å utføre programvarens funksjonalitet. For en bedre brukeropplevelse ble Jupyter Notebook integrert i brukergrensesnittet.

3.4 Voilà

Voilà er et verktøy som kan gjøre Jupyter Notebook om til en frittstående nettapplikasjon med støtte for interaktive dashbord[26]. Verktøyet gjør det også mulig å kjøre gjennom alle cellene i en Jupyter Notebook-fil med et tastetrykk. Voilà ble implementert for å kunne å tilby en raskere oppstart av Open AR Sandbox.



Figur 12: Utføringsmodellen til Voilà[26]

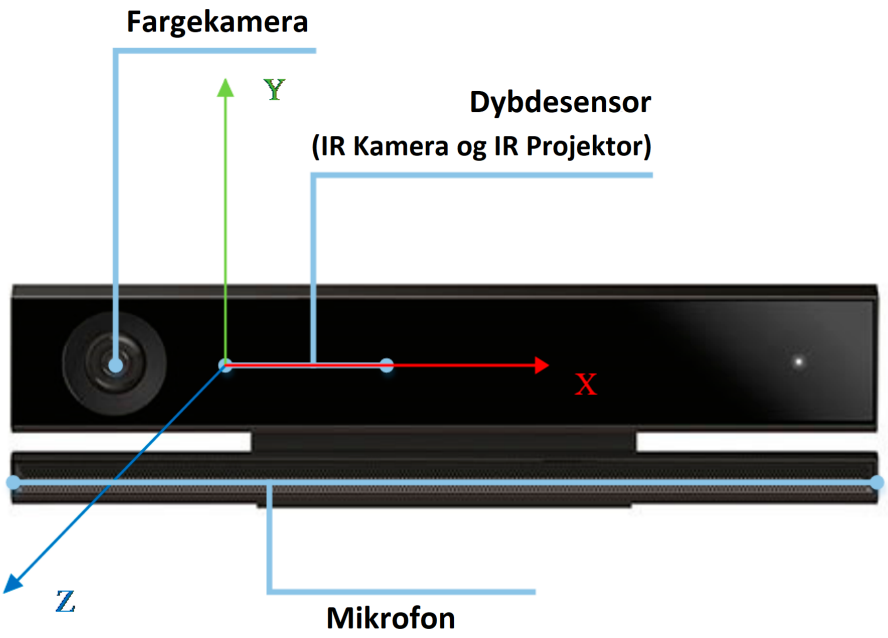
Utføringsmodellen i figur 12 viser hvordan Voilà fungerer med backenden representert på venstre side, og frontenden på høyre side. Når man skriver inn Jupyter Notebook-URL-en i nettleseren, sendes det en HTTP-forespørsel til serveren, som åpner notebooken og kjører en assosiert kernel. Så kjører Voilà alle notebook-cellene

og fyller inn cellenes utdata. Den legger også til en Voilà mal for å konvertere notebooken til et HTML-dokument som sendes over til frontenden, hvor dokumentet blir gjengitt. Kernelen forblir kjørende i backenden. HTML inneholder også JavaScript kode som åpner en WebSocket-tilkobling til kernelen i backenden. Denne mekanismen tillater widgetene i frontenden til å kjøre koden i backenden[26], som vist i figur 12.

3.5 Xbox One Kinect Sensor v2

Xbox One Kinect sensor v2 er et bevegelsesfølende kamera produsert av Microsoft i 2013 som er utviklet for "Xbox One". Kameraet har siden fått støtte for operativsystemet Windows ved bruk av en USB adapter. Dette har gjort det mulig å bruke Kinect v2-sensoren sammen med Open AR Sandbox.

Sensoren består av et fargekamera med 1920x1080 oppløsning og et dybdekamera med 514x424 oppløsning. Kameraet har også en innebygget infrarød sensor, noe som gjør at den kan brukes i et mørkt rom uten noen form for synlig lys[27]. En fullstendig oversikt over kameraets funksjonalitet vises i figur 13.



Figur 13: Xbox One Kinect Sensor v2[28]

3.6 BenQ MW632ST projektor

Projektoren som brukes til å projisere ned i sandkassen er en BenQ MW632ST. Projektoren kan vise opptil 64 tommer fra 1 meters avstand, og passer derfor godt til sandkassen dimensjoner. I tillegg har den et høyt kontrastforhold på 13000:1. Det vil si at et hvitt bilde kan bli 13000 ganger lysere enn det mørkeste bildet. Dette resulterer i et klart bilde av fargekonturene i sanden.

3.7 GitHub

GitHub er en nettbasert platform for oppbevaring og lagring av kode og forskjellige prosjekter. GitHub tilbyr en enkel og effektiv måte å programmere sammen med flere personer. Alle versjoner av et prosjekt blir lagret på serveren til GitHub, noe som gjør at alle som deltar i prosjektet har tilgang til alle versjonene av prosjektet. Takket være dette kan alle som er med i prosjektet laste opp sitt eget arbeid før det blir kombinert med selve prosjektet[29].

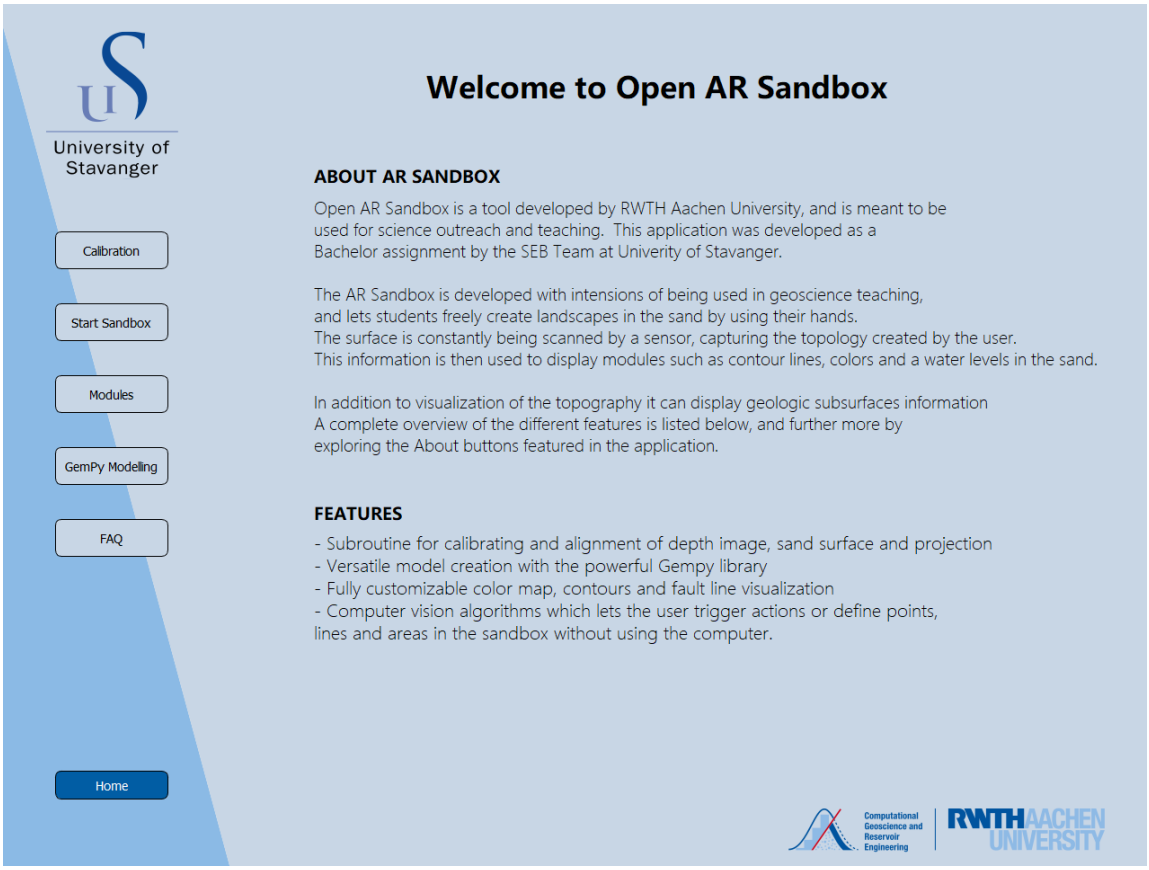
GitHub ble brukt for å lagre og distribuere kode blant medlemmene i gruppen.

4 Oppbygning og konstruksjon

Dette kapittelet tar for seg utviklingen av det nye brukergrensesnittet til Open AR Sandbox og utviklingen av GemPy modelleringsmodulen.

4.1 Open AR Sandbox brukergrensesnitt

Formålet med brukergrensesnittet var at det skulle være enkelt å bruke for både norske og internasjonale studenter, uten noen forkunnskaper i Python. For å oppnå dette ble det lagt fokus på brukervennlighet gjennom hele utviklingsprosessen. Med brukervennlighet menes det at brukergrensesnittet skal være enkelt å operere selv for uerfarne brukere, samt at brukere skal oppnå ønsket resultat med færrest mulig tastetrykk. Brukergrensesnittet har blitt skrevet på engelsk, slik at det skal være tilpasset internasjonale studenter. Hjemskjermen er vist i figur 14.



Figur 14: Hjemskjermen som vises ved oppstart

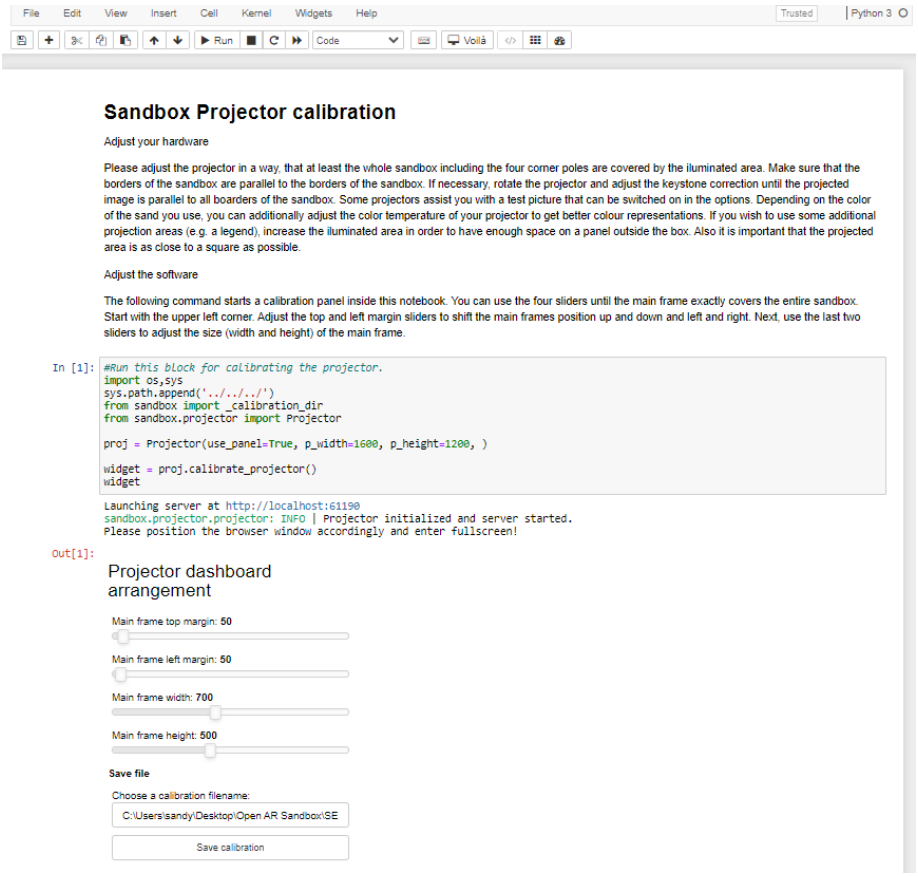
Generelt om brukergrensesnittet
Brukergrensesnittet består av en sidemeny, to logoer og et widget-område. Hvordan widget-området defineres blir forklart i delkapittel 4.1.5, paragraf "QStackedWidget". For å fremme utviklerene og vise til kildekode vises de to logoene på alle menyvalgene. Logoene er hyperlinker til GitHub-sidene til utviklerene av Open AR Sandbox og det nye brukergrensesnittet/Gempy-modelleringsmodulen. UiS logoen viser til utviklerene ved UiS[30], mens RWTH logoen viser til utviklerene ved RWTH[1]. I applikasjonens brukergrensesnitt er det lagt til flere infoknapper, samt en egen "FAQ" side som referer til en brukermanual. Brukermanualen er laget for det nye brukergrensesnittet til Open AR Sandbox og Gempy-modeleringsmodulen.

Åpning av applikasjonen
For åpning av applikasjonen har det blitt lagt til en snarvei på skrivebordet for å enkelt kunne starte applikasjonen og serveren. Når denne snarveien blir brukt, vil det bli åpnet to vinduer. Det ene er selve brukergrensesnittet, mens det andre er et terminalvindu. Terminalvinduet åpnes i bakgrunnen og kjører så lenge applikasjonen er åpen. I terminalvinduet vil man kunne se at det blir åpnet en Jupyter Notebook-server i det applikasjonen starter opp. Ved hjelp av terminalvinduet er det også mulig og overvåke hvilke kommandoer som kjøres når Jupyter Notebook-serveren er åpen og eventuelle feilkoder som kan forekomme. Snarveien er en BAT-fil som blir forklart videre i kapittel 5.2.

4.1.1 Utvikling

Koden til brukergrensesnittet ble utført i to respektive filer. En Python-fil for å utføre funksjonaliteten til applikasjonen, og en CSS-fil for de visuelle aspektene. Sammen danner disse brukergrensesnittet.

Modifisering av Jupyter Notebook-filer
Funksjonaliteten til modulene kjøres gjennom Jupyter Notebook-filer. Disse Jupyter Notebook-filene ligger lokalt på maskinen og åpnes direkte i brukergrensesnittet. Det er tatt høyde for at dette brukergrensesnittet skal brukes av studenter uten forkunnskaper innen programmering. Det har derfor blitt gjort endringer på de originale Jupyter Notebook-filene slik at de skal være tilpasset til applikasjonen. Filene er blitt tilpasset ved at unødvendige celler er fjernet slik at Jupyter Notebook-filene kun består av 1-3 celler. I tillegg er det lagt til forklaringer for hva hver av de ulike cellene gjør.



Figur 15: Modifisert Jupyter Notebook-fil

I figur 15 er det vist en modifisert Jupyter Notebook-fil, som i dette tilfellet er kalibrasjonsfilen til projektoren. Denne filen har blitt endret på slik at det kun er nødvending å kjøre én celle for å starte kalibrering av projektoren. Det har også blitt fjernet muligheten for redigering og sletting av celler i Jupyter Notebook-filene mens applikasjonen kjører. Dette har blitt gjort ved å endre "metadataen" til hver enkelt Jupyter Notebook-fil. Metadata er data som gir informasjon om annen datas innhold[31], noe som gjør det mulig å konfigurere cellene til en Jupyter Notebook-fil.

4.1.2 Oppstart av Jupyter Notebook server

For å starte Jupyter Notebook må man vanligvis bruke terminalvinduet. Dette ville brutt med oppgavens formål om å være brukervennlig for geologistudentene. Dette ble løst ved å kjøre terminalkommandoen direkte i Python, ved bruk av klassene "popen" og "chdir" i Python-modulen "os"[32]. "Chdir" spesifiserer filstien til Jupyter Notebook-filene slik at operativsystems-avhengige kommandoer kan utføres fra filstien. For å kommunisere med terminalvinduet brukes "popen". "Popen" fungerer som

en kommunikasjonslink mellom Python og operativsystemetet.

Funksjonaliteten til Open AR Sandbox er avhengig av at Jupyter Notebook-serveren kjøres fra start. Derfor startes Jupyter Notebook-serveren i bakgrunnen i det man åpner applikasjonen som vist i kode 1.

```
    def __init__(self):
        os.chdir(r"C:\Users\sandy\Desktop\SEB_Addon\mainGUI\notebooks\
                                                tutorials")
        self.stream = os.popen('jupyter notebook')
```

Kode 1: Åpning av Jupyter Notebook

Ved at serveren kjøres i bakgrunnen mens applikasjonen brukes, kan moduler åpnes og lukkes uten at applikasjonen må startes på nytt.

4.1.3 Avslutte Jupyter Notebook Server

Ved oppstart av applikasjonen vil serveren starte på port 8888 som vist i figur 3. Serveren vil kjøre helt til applikasjonen lukkes. Når applikasjonen lukkes er det essensielt at serveren også lukkes, hvis ikke vil denne porten være opptatt neste gang man starter applikasjonen. Vanligvis avsluttes en Jupyter Notebook-server med tastetrykkene "Control+C" i terminalvinduet. Når man lukker applikasjonen avsluttes Jupyter Notebook-serveren ved bruk av "os" og "signal", som vist i kode 2. "Signal" gjør det mulig å sende egendefinerte tastaturavbrudd, samt håndtere tastaturavbruddene[33]. os.kill() og os.getpid() stopper den nåværende prosessen.

```
    def closeEvent(self,event):
        reply = QMessageBox.question(self, 'Window Close', 'Do you want to
                                                close all Jupyter Notebook
                                                ports?',
                QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
        if reply == QMessageBox.Yes:
            if hasattr(signal, 'CTRL_C_EVENT'):
                os.kill(os.getpid(), signal.CTRL_C_EVENT)
            time.sleep(0.1)
            event.accept()
        else:
            event.ignore()
```

Kode 2: Avslutte Jupyter Notebook-server

4.1.4 Bruken av Voilà i brukergrensenittet

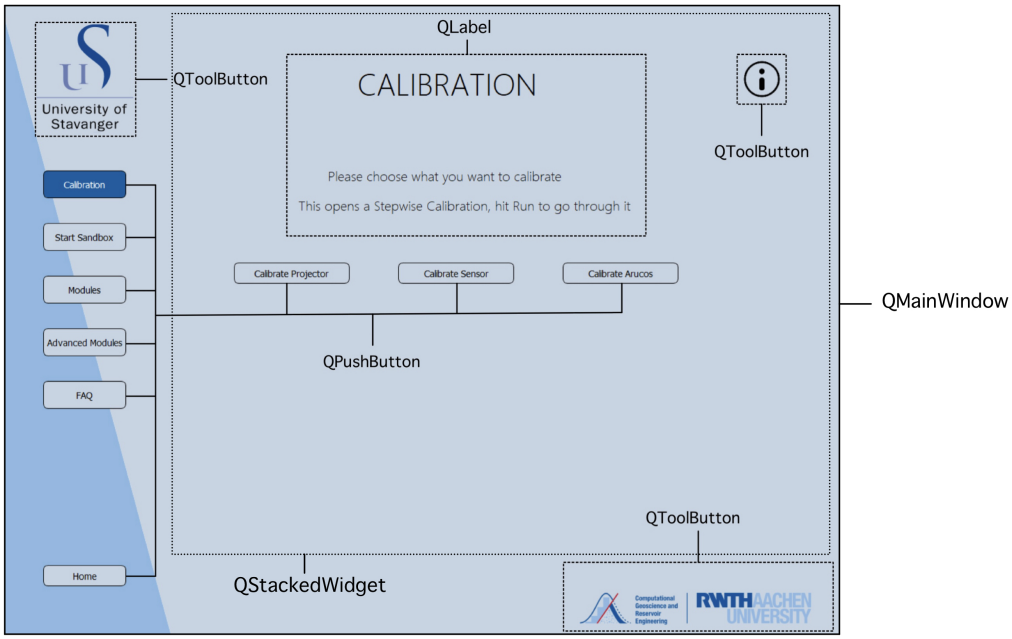
I brukergrensesnittet brukes Voilà til å kjøre gjennom de Jupyter Notebook-filene som ikke har interaktive widgets. Filer med interaktive widgets har ikke støtte for Voilà. For å kjøre en Jupyter Notebook-fil ved bruk av Voilà-kommandoen, må man først spesifisere filstien til mappen hvor filen ligger. I kode 3 brukes os-modulen til Python for å endre filstien til mappen med de ulike modulene i applikasjonen. Deretter brukes metoden "popen" til å kjøre Voilà gjennom terminalen med kommandoen "voila {filnavn}".

```
    def advanced_startup_voila(self, combobox):
        os.chdir(r"C:\Users\sandy\Desktop\SEB_Addon\mainGUI\notebooks\
                                                tutorials")
        stream = os.popen('voila with_a.ipynb')
```

Kode 3: Voilà-kommandoen brukes til åpne en Jupyter-fil

4.1.5 Oppbyggningen av brukergrensesnittet med PyQt

I dette kapittelet forklares de ulike PyQt-klassene som ble brukt til utvikling av brukergrensesnittet.



Figur 16: Brukergrensesnittet med PyQt-klasser

QMainWindow

QMainWindow er grunnmuren til brukergrensesnittet. Sammen med sine relaterte klasser danner QMainWindow-klassen rammerverket til applikasjonen[34]. Geometrien til QMainWindow dekker hele brukergrensesnittet som vist i figur 16.

```
class MainWindow(Qtw.QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.setGeometry(40, 150, 1185, 800)
        self.setWindowTitle('Open AR Sandbox')
        self.home()
        self.default_screen_widget()
        self.calibration_widget()
        self.modules_widget()
        self.modelling_tool_widget()
        self.start_up_widget()
        self.faq_widget()
        sshFile="sandboxStyle.css"
        with open(sshFile,"r") as fh:
           self.setStyleSheet(fh.read())
        os.chdir(r"C:\Users\sandy\Desktop\SEB_Addon\mainGUI\notebooks\
                                                tutorials")
        self.stream = os.popen('jupyter notebook')
```

Kode 4: MainWindow som initieres ved oppstart

MainWindow er programmets hovedklasse. Den består av en init-funksjon som initieres ved oppstart. I init-funksjonen settes geometrien til brukergrensesnittet, vindustittelen, og alle widgetene som brukes blir definert. Uten å initiere dette ved start vil ikke brukergrensesnittet vises. Til slutt tas CSS-stilarket inn, hvilket definerer utseendet til brukergrensesnittet, som vist i kode 4.

QPushButton

QPushButton er den mest brukte PyQt-klassen i brukergrensesnittet. Denne PyQt-klassen ble brukt til å gi knappene funksjonalitet. I applikasjonen er disse knappene koblet til kommandoer som utføres av datamaskinen ved museklikk. Eksempelvis består sidemenyen av knapper definert med QPushButton-klassen.

QWidget

QWidget er kjernen til brukergrensesnittet. Widgeten mottar museklikk, tastaturtrykk og andre hendelser fra vindusystemet, og danner en representasjon av seg selv på skjermen. En widget er alltid rektangulær og er sortert i Z-aksen[35]. Dette betyr at en widget kan vises foran en annen widget.

QStackedWidget
QStackedWidget brukes for å stable flere widgeter på hverandre. De ulike widgetene er koblet til sine respektive knapper i sidemenyen. Slik kan man navigere mellom forskjellige widgeter i samme QMainWindow.

Hvert menyvalg i applikasjonen viser en egen widget. Widgetene bruker samme område på skjermen, men viser forskjellig informasjon ettersom widgetene er koblet sammen i stabler. Sammenkoblingen av widgetene ble gjort ved bruk av PyQt-klassen QStackedWidget. De stablede widgetene ligger i widgetområdet, men kun én widget vises av gangen. I figur 16 vises kalibrerings-widgeten som et eksempel.

Hvordan QPushButton, QWidget og QStackedWidget fungerer sammen

For å forklare hvordan QPushButton, QWidget og QStackedWidget jobber sammen for å bytte mellom widgeter, vil metodene "home" og "set_screen_widget" nå bli gått gjennom med Calibration-knappen som eksempel.

```
    def home(self):
        # Adding buttons to main window
        calibration_btn = Qtw.QPushButton('Calibration', self)
        calibration_btn.setGeometry(Qtc.QRect(60, 250, 130, 50))
        calibration_btn.clicked.connect(lambda: self.set_screen_widget(self
                                                .menu_sw, self.calibration_w,
                                                 calibration_btn))

        # Define all widgets
        self.default_screen_w = Qtw.QWidget(self)
        self.calibration_w = Qtw.QWidget(self)

        # Adding stacked widget
        self.menu_sw = Qtw.QStackedWidget(self)
        self.menu_sw.setGeometry(Qtc.QRect(250, 40, 1500, 1500))
        self.menu_sw.addWidget(self.calibration_w)
```

Kode 5: Funksjon for hjemskjermen

I metoden "home", som vises i kode 5, blir Calibration-knappen først definert som en QPushButton. Deretter brukes klassen "setGeometry" til å definere hvor på skjermen knappen skal ligge, samt hvor stor den skal være (x-start, y-start, x-bredde, y-høyde). Så kobles kalibrerings-knappen til kalibrerings-widgeten via set_screen_widget-metoden. Set_screen_widget-metoden velger hvilken widget som skal vises ut ifra hvilken menyknapp som er valgt.

Set_screen_widget tar inn hvilken stacked_widget som skal endres, widgeten som skal vises og hvilken knapp som er trykket på. Knappen som er trykket på får en annen farge i brukergrensesnittet slik at det er mulig å se hvilken meny brukeren er i. I metoden brukes QStackedWidget-egenskapen "setCurrentWidget" til å ta inn

widgeten som skal vises. Den forrige menyknappen som er blitt brukt blir så endret tilbake til den orginale CSS-en, mens den nye menyknappen får en ny CSS.

```
    def set_screen_widget(self, stacked_widget, widget, button=None):
        stacked_widget.setCurrentWidget(widget)
        if button:
            if self.last_button_clicked:
                qss = """Ny CSS kode for knapp"""
                last_button = self.last_button_clicked.pop()
                last_button.setStyleSheet(qss)
            qss = """Orginal CSS kode for knapp"""
            button.setStyleSheet(qss)
            self.last_button_clicked.append(button)
```

Kode 6: Metode for endring av widget i brukergrensesnittet

Videre i kode 5 blir widgetene definert ved bruk av QWidget. Widgetene må defineres for å være funksjonelle i brukergrensesnittet. Til slutt legges kalibreringswidgeten til som en av de stablede widgetene. Dette ble gjort for samtlige widgeter, på lik måte som vist i kode 5.

QDialog
QDialog er en widget som ligger øverst i Z-aksen til brukergrensesnittet. Det som skiller QDialog-widgeten fra en QWidget er at QDialog-widgeten åpnes i et eget vindu[36]. Dialogene i brukergrensesnittet brukes blant annet til å vise informasjon om de forskjellige modulene. Hvordan dette ser ut i brukergrensesnittet er vist i figur 24b.

QToolButton
QToolButton gjør det mulig å legge til knappefunksjonalitet til for eksempel ikoner. Forskjellen på en QPushButton og en QToolButton er at sistnevnte hovedsaklig brukes til ikoner, mens QPushButton brukes til knapper med tekst[37]. I applikasjonen brukes QToolButton til tre handlinger:

1. Åpne GitHub-siden til utviklere ved UiS og RWTH i nettleseren
2. Åpne info-dialoger i brukergrensesnittet
3. Åpne programvaren til Open AR Sandbox via Jupyter Notebook i brukergrensesnittet

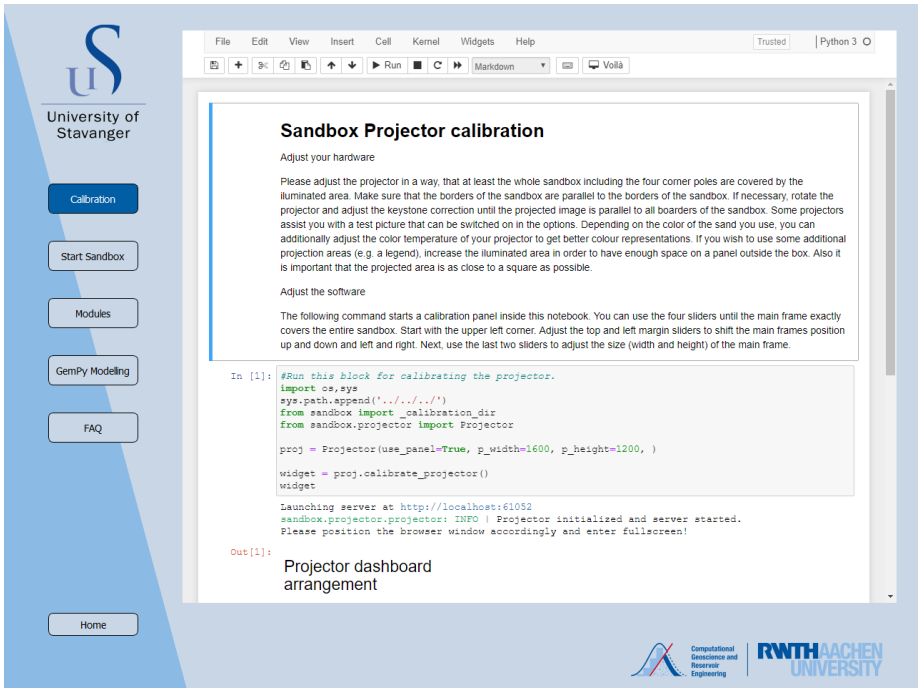
QLabel
QLabel blir brukt for å vise tekst i brukergrensesnittet.

QWebEngineView

For å gjøre Jupyter Notebook-serveren tilgjengelig i applikasjonen ble QWebEngineView-klassen brukt. QWebEngineView er en PyQt-klasse som gjør det mulig å implementere en nettside i brukergrensesnittet[38], som vist i figur 17. Hver av de opprinnelige Jupyter Notebook-filene til Open AR Sandbox, ble tidligere kjørt i nettleseren. Disse filene blir nå åpnet i brukergrensesnittet via QWebEngineView. Metoden definerer en ny QWebEngineView og åpner URL-en til Jupyter Notebook-filen i den integrerte nettleseren (se kode 7).

```
    def calibrate_projector_dialog(self):
        self.browser = QWebEngineView(self.jupyter_w)
        self.browser.setUrl(QUrl("http://localhost:8888/notebooks/
                                  00_Calibration/
                                  1_calib_projector.ipynb"))
        self.browser.setGeometry(Qtc.QRect(0,0,1000, 800))
        self.set_screen_widget(self.menu_sw, self.jupyter_w)
```

Kode 7: Åpning kalibreringsmodul



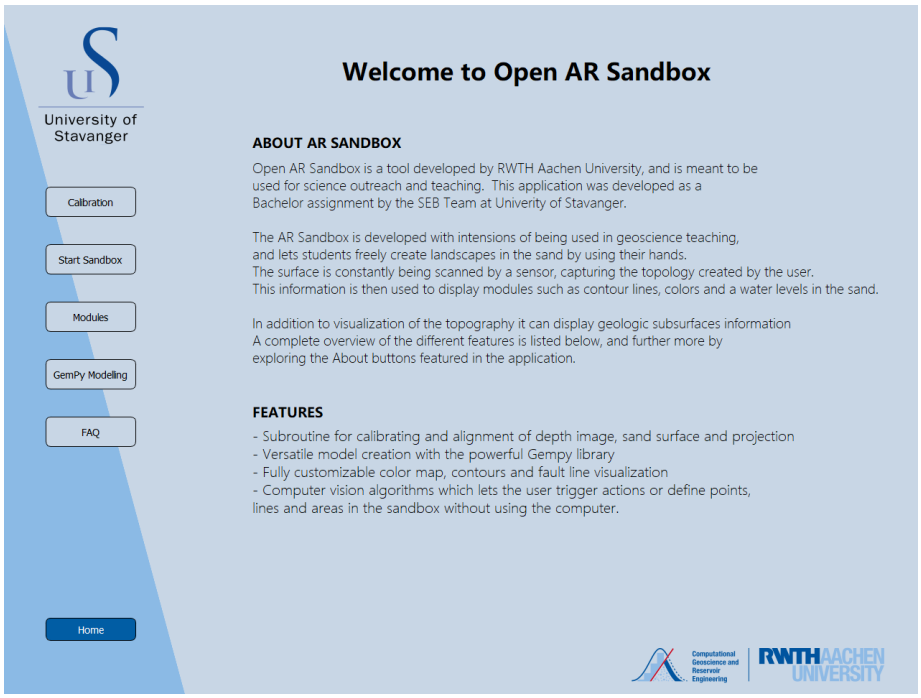
Figur 17: Jupyter Notebook-fil i brukergrensesnittet

4.2 Gjennomgang av menyvalg i brukergrensesnittet

I dette kapittelet vil funksjonaliteten til de ulike menyvalgene bli forklart.

4.2.1 Hjemskjerm

Hjemskjermen er det første brukeren ser når applikasjonen startes. Den består av en introduksjon til Open AR Sandbox, hvem som har utviklet det og hvilke funksjonaliteter som støttes. Hver widget inneholder informasjonsknapper med forklaringer på hvordan man bruker widgetens funksjonaliteter.



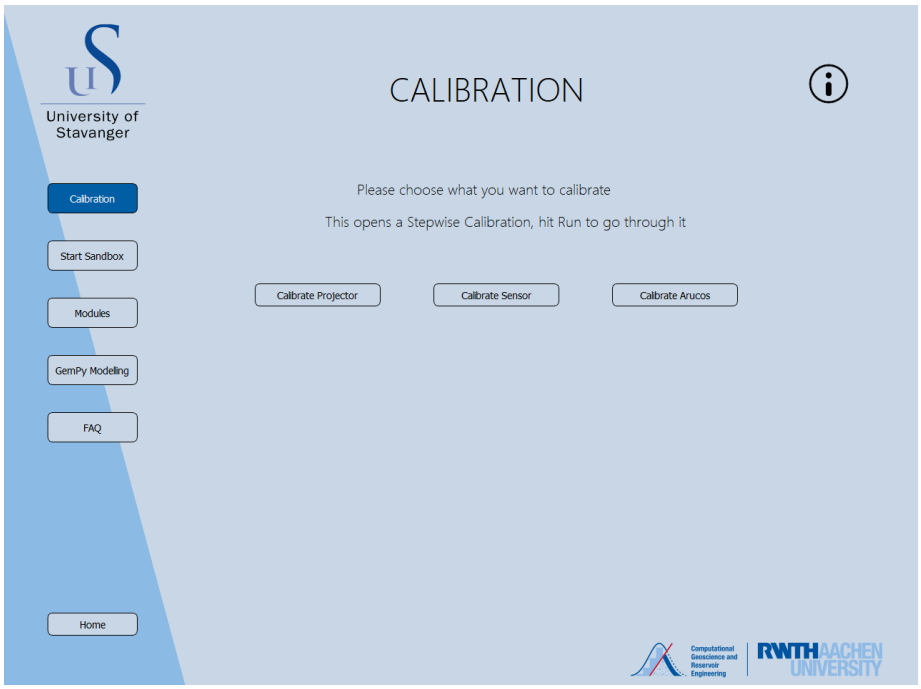
Figur 18: Hjemskjerm

4.2.2 Kalibrering

I delmenyen "Calibration" er det mulig å kalibrere projektor og sensor, samt bruke ArUco til å kontrollere om kalibreringen er korrekt. Knappene åpner egne Jupyter Notebook-vinduer i brukergrensesnittet. Deretter kan bruker gå stegvis gjennom Jupyter Notebook-filen ved å klikke "Run" for hver Jupyter Notebook-celle.

Kalibreringen er en essensiell del i bruken av Open AR Sandbox, ettersom feil kalibrering vil gi feilprojiserte bilder i sandkassen. Det er derfor viktig at brukere av

systemet gjør seg godt kjent med kalibreringen. Etter brukeren har kalibrert projektor og sensor, lagres kalibreringsfilene automatisk til neste gang man skal brukes programvaren. Forøvrig anbefales det å kalibrere projektor og sensor før hver oppstart av programvaren, for å unngå feilprojiserte bilder.



Figur 19: Kalibrasjonsmeny

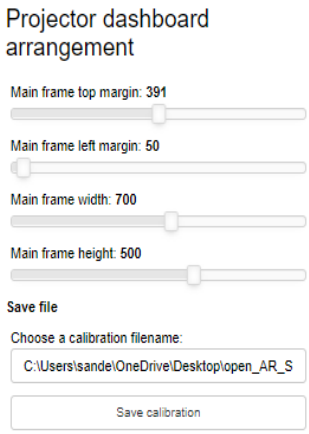
Generelt om kalibrering

For å starte kalibreringen klikker man på en av kalibreringsknappene, som vises i figur 19. Kalibreringen av sensor og projektor foregår ved bruk av to vinduer. Det ene vinduet viser kalibreringsmodulen i brukergrensesnittet, mens det andre viser bildet som skal projiseres i sandkassen. Bildet som skal projiseres i sandkassen åpnes automatisk i nettleseren ved å kjøre gjennom første Jupyter Notebook-celle i filen. Dette vinduet må flyttes over til projektoren og åpnes i fullskjermmodus.

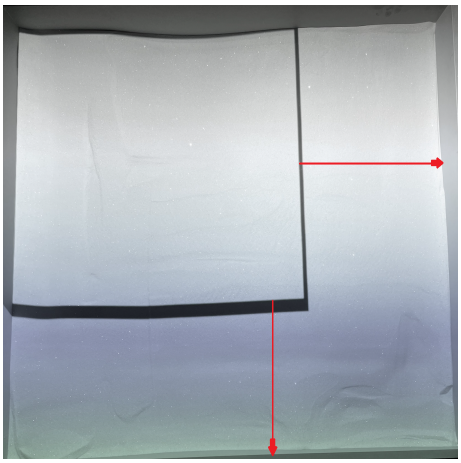
Kalibrering av projektor

I projektorkalibreringen er målet å få hele projektorbildet innenfor rammene i sandkassen, som vist i figur 20b. For å oppnå dette må projektorbildet først være parallelt med sandkassens lengde og bredde. Om nødvendig, må man justere "Keystone-verdiene" på projektoren for å få lengdene parallelle. Etter dette må bredden, lengden, toppmargin og bunnmargin endres på ved å bruke skyvebryterne i kalibrasjonspanelet. Disse verdiene er målt i millimeter.

Når kalibreringen er fullført kan man lagre kalibreringsfilen for senere bruk. Hvordan kalibrasjonspanelet til projektoren er satt opp er vist i figur 20a.



(a) Kalibrasjonspanel for projektor



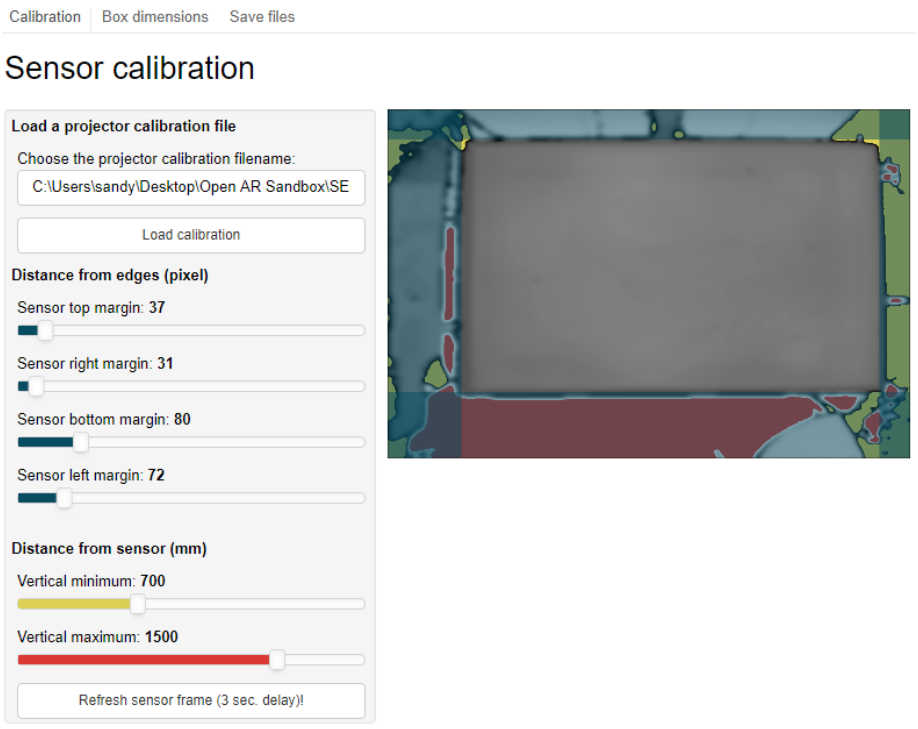
(b) Projektorbilde i sandkassen

Figur 20: Kalibrering av projektor

Kalibrering av sensor
Sensorkalibreringen er mer komplisert enn projektorkalibreringen. I tillegg til å kalibrere de horisontale verdiene må også de vertikale dybdebilde kalibreres. Dette gjøres for å unngå uønskede rekalkulasjoner når man beveger hånden over bildet som projiseres i sandkassen.

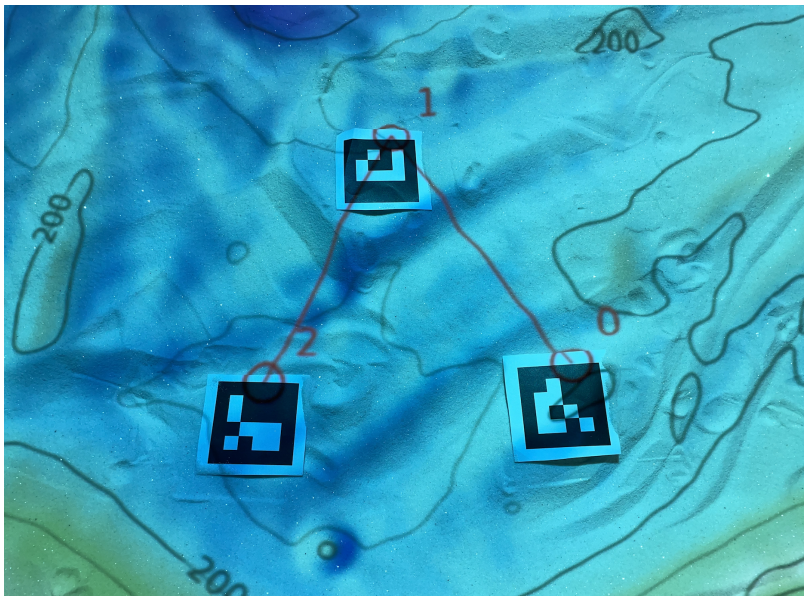
Det er viktig at sensoren er korrekt plassert over sandkassen. Kinect-sensoren skal posisjoneres slik at den er parallell med sandkassens overflate.

Kalibreringen foregår i Jupyter Notebook-filen ved bruk av skyvebrytere. Venstre-, høyre-, topp- og bunn-margin må endres slik at sensoren ikke leser utsiden av sandkassen. Etter dette må vertikal minimum- og maksimum-distanse endres på. Verdiene er målt i millimeter. Etter kalibreringen kan kalibreringsfilen lagres.



Figur 21: Kalibrasjonspanel for sensor

Verifisering av kalibrering med ArUco-merker
Formålet med denne modulen er at brukeren skal kunne verifisere at sensor- og projektor-kalibreringen er korrekt. Dette gjøres ved bruk av ArUco-merker, som plasseres i sandkassen. Om kalibreringen stemmer vil de bli oppdaget av sensoren og en sirkel sammen med ID vil vises på toppen av merket, som vist i figur 22. ArUco-merker brukes også i andre moduler for å simulere jordskred, elektroder eller vann. Merkene vil da definere områdene simulasjonene skal starte.

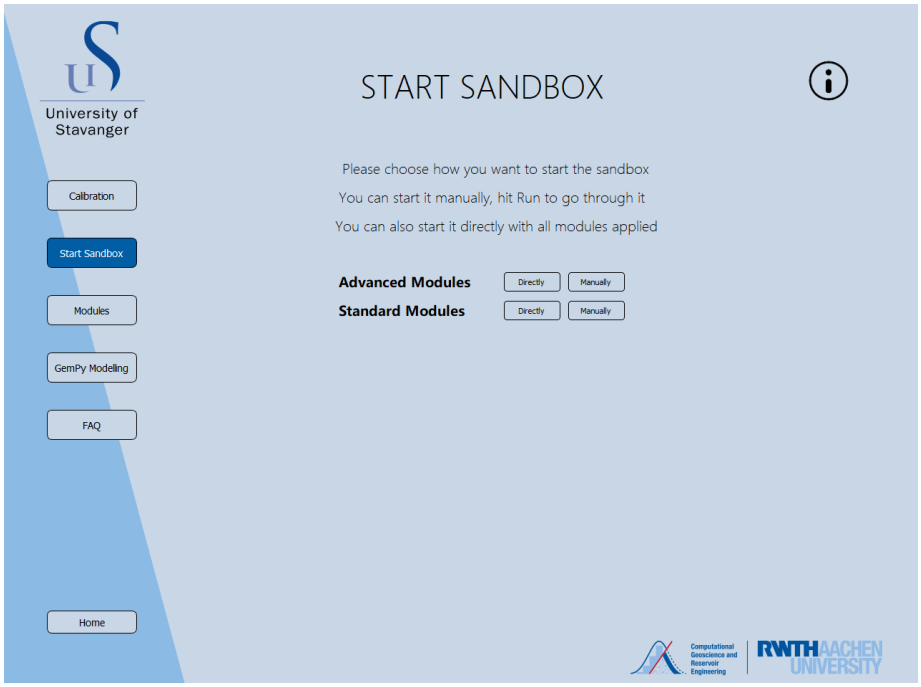


Figur 22: ArUco-merker med ID-nummer

4.2.3 Oppstart av server

I Start Sandbox-delmenyen er det mulig å kjøre flere moduler samtidig ved bruk av Open AR Sandbox-serveren. Serveren kan startes manuelt ved å kjøre gjennom hver Jupyter Notebook-celle eller automatisk ved bruk av protokollen Voilà. Denne automatiske egenskapen er lagt til for å tilby en raskere og enklere oppstart av serveren. Det er også blitt konstruert en informasjonsknapp i delmenyen slik at det skal være lett for nye brukere og kunne velge hvilken oppstartsvalg som passer best med deres formål.

Serveren kan starte enten forhåndsdefinerte standardmoduler eller forhåndsdefinerte avanserte moduler.



Figur 23: Oppstartsvalg med informasjonsknapp øverst til høyre

Standardmoduler
Ved valg av standardmoduler blir følgende moduler initialisert i serveren, og gjort tilgjengelig i kontrollpanelet.

- TopoModule
- GradientModule
- LoadSaveTopoModule
- LandslideSimulation
- SearchMethodsModule
- CmapModule
- ContourLinesModule

Kontrollpanelet vises i figur 5. I kontrollpanelet kan brukeren velge hvilken modul som skal være aktiv og endre på innstillengene til de aktive modulen.

Avanserte moduler
Ved valg av avanserte moduler vil standardmodulene initialiseres sammen med følgende avanserte moduler.

- GemPyModule
- LandscapeModule
- GeoelelectrisModule

Oppstart av server har blitt delt inn i to forskjellige oppstartsvalg på grunn av at de avanserte modulene bruker mye prosessorkraft og bruker lang tid for å initialiseres. Ettersom hver av de avanserte modulene krever mye prosessorkraft, bør disse kjøres hver for seg.

Manuell og direkte oppstart av Open AR Sandbox
Når serveren startes manuelt med avanserte moduler eller standardmoduler, åpnes et nettleservindu i applikasjonen ved bruk av QWebEngineView. Deretter brukes URL-linken til den korresponderende Jupyter Notebook-filen som skal åpnes til å sende en HTTP-forespørsel til Jupyter Notebook serveren. Som nevnt tidligere har denne serveren kjørt i bakgrunnen siden applikasjonen ble startet. Notebook-filene som kjøres i applikasjonen ligger lokalt på maskinen og er derfor spesifisert med URL-en `localhost:8888/{filsti}` som vist i kode 7.

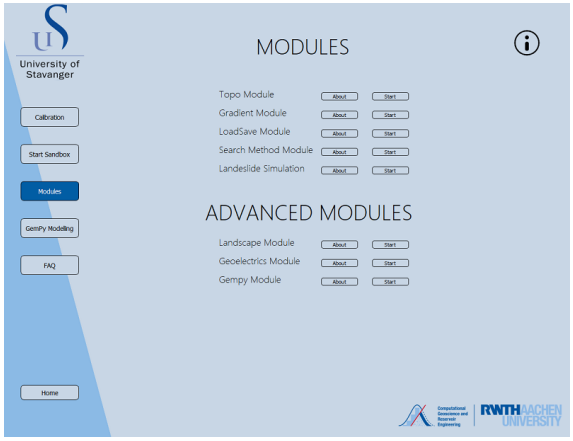
```
    def advanced_startup_voila(self):
        os.chdir(r"C:\Users\sandy\Desktop\SEB_Addon\mainGUI\notebooks\
                                                tutorials")
        stream = os.popen('voila with_a.ipynb')
```

Kode 8: Oppstart av server med avanserte moduler ved bruk av Voilà

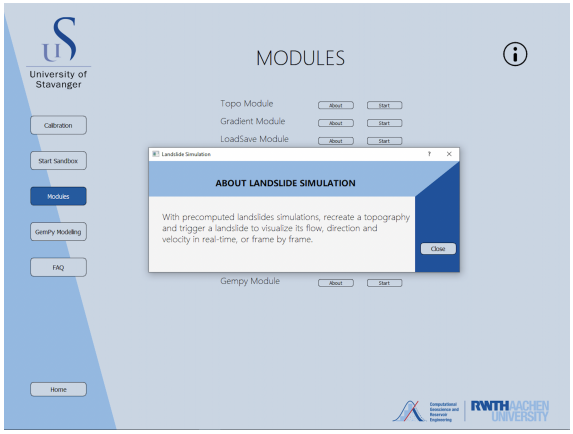
Når serveren startes direkte ved bruk av Voilà brukes "popen" åpnes Jupyter Notebook-filen i en Voilà-server. Dette gjøres med kommandoen "voila filnavn" som vist i kode 8. Et nettleservindu med Jupyter Notebook-filen vil åpnes fra Voilà-serveren. Dette vinduet viser antall celler som må kjøres gjennom og hvor langt denne prosessen har kommet. Deretter vil serveren starte som normalt ved å åpne kontrollpanelet i et vindu og bildet som skal projiseres i sandkassen i et annet vindu.

4.2.4 Moduler

I delmenyen "Modules" er det mulig starte enkeltmoduler. Dette gjøres intuitivt ved å klikke "Start" som åpner en Jupyter Notebook-fil i brukergrensesnittet. Oppstart av modulene er forklart i kapittel 4.1.5 under paragrafet "QWebEngineView".



(a) Modulsiden



(b) Informasjonsvinduet til Landslide-modulen

Figur 24: Modulside med og uten informajonsknapp aktivert

For hver modul er det lagt til en kortfattet forklaring på engelsk som vist i figur 24b.

4.2.5 FAQ

I delmenyen "FAQ" er det laget en brukermanual i PDF-format som åpnes i en nettleser slik at den kan brukes underveis. Denne brukermanualen er skrevet på engelsk og inneholder følgende informasjon:

- Informasjon om de forskjellige delmenyene i brukergrensesnittet
- Hvordan en bruker skal starte programmet
- Manual for kalibrering av projektor og sensor, samt testing av kalibreringen ved bruk av ArUco-merker.
- Manual for oppstart av forskjellige moduler.
- Manual for å konstruere modeller i GemPy modelleringsmodulen, samt projisere disse modellene i sandkassen.
- Manual for oppstart av serveren.
- Manual for installasjon av programvare

4.2.6 Projisere GemPy-modell i sandkassen

Delmenyen "GemPy Modelling Module" består av to knapper. Den ene knappen brukes til å åpne GemPy-modelleringsmodulen som ble utviklet under oppgaven. Den andre knappen brukes til å projisere den konstruerte Gempy-modellen ned i sandkassen.

Når modellen er ferdigkonstruert i GemPy-modelleringsmodulen må brukeren laste modellen opp til sandkassen. Modellen vil bli lagret som en pickle-fil i mappen som innenholder Jupyter Notebook-filene til GemPy-modulen. Når filen er lagret kan den bli lastet opp til sandkassen gjennom en Jupyter Notebook-fil som laster inn pickle-filen til Open AR Sandbox sin GemPy modul. For mer informasjon om GemPy-modelleringsmodulen, og hvordan denne ble utviklet, se delkapittel 4.3.

Figur 25: GemPy-modell projisert i sandkassen

Figur 25 viser den projiserte GemPy-modellen fra figur 34.

4.3 GemPy Modelleringsmodul

Formålet med GemPy-modelleringsmodulen er at brukere skal kunne lage modeller i GemPy uten forkunnskaper i Python. Modulen kan brukes som en frittstående programvare eller som en modul til Open AR Sandbox gjennom det nye brukergrensesnittet. Når en modell er konstruert i modulen kan den projiseres i sandkassen fra det nye brukergrensesnittet til Open AR Sandbox.

4.3.1 Grunnleggende GemPy

Gempy kan deles inn i tre hovedelementer: Flater, serier og kompilator. Sammen danner disse elementene grunnmuren til GemPy.

Flater
En flate i GemPy representerer et geologisk lag. Et geologisk lag er et lag med naturlig forekomst av mineraler eller løsmasse som er formet av gravitasjon og andre geologiske lag over lengre tid. I GemPy blir flater representert av punkter i et koordinatsystem og blir fremstilt som et plan mellom disse punktene. For å plotte en flate i GemPy må det defineres minimum tre punkter og en orientering til flaten. En orientering representerer normalen til flaten i et spesifikt punkt i koordinatsystemet og defineres enten med en vektor eller retningsvinkel(Azimuth) og fallvinkel(Dip) verdier(se figur 29).

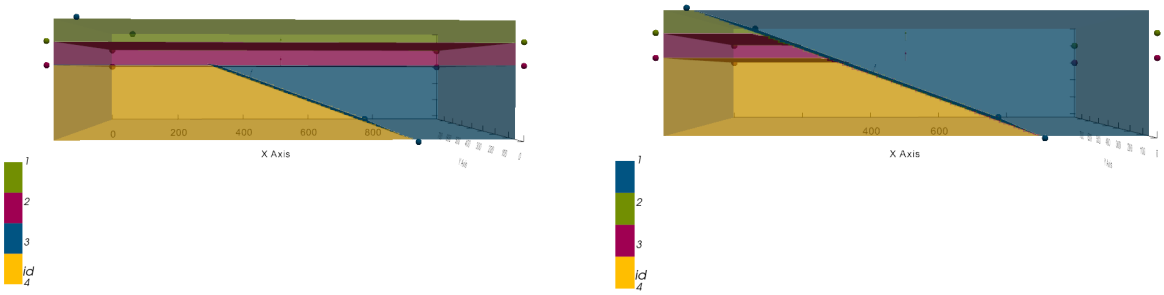
Koordinatpunkter og orienteringer utgjør sammen de to hovedpunktene til algoritmen som blir brukt av GemPy. Denne algoritmen er en interpolasjonsalgoritme som bruker koordinatpunkt og orienteringer for å interpolere et 3D-skalarfelt i modellen[39]. Disse skalarfeltene brukes av GemPy for å kalkulere hvordan en flate påvirkes av en annen flate.

Flater i GemPy blir visualisert fra og med starthøyden (negative z-koordinater) til og med neste flate. Dersom det ikke finnes noen flate over nåværende flate blir den visualisert til og med toppen av modellen. Det er derfor viktig at det blir definert en "kjellerflate" i bunnen av hver modell. Kjellerflaten gjør at modellen blir definert med en farge fra laveste punkt i koordinatsystemet.

Serier
For at en flate skal kunne plottes inn i koordinatsystemet må den være en del av en serie. En serie er en gruppe med flere flater, hvor flatene påvirkes av hverandre. I GemPy blir serier plassert i en kronologisk rekkefølge når de blir konstruert. Når seriene skal plottes vil seriene prioriteres ut ifra den kronologiske rekkefølgen. Denne egenskapen gjør det mulig å modellere forkastning og diskordans i GemPy.

Forkastning er et brudd som kan forekomme i geologiske lag på grunn av jordskjelv. Et jordskjelv kan føre til at det blir dannet grenser i ett eller flere geologiske lag som brått beveger seg i forhold til hverandre[40]. Forkastninger i GemPy må ligge først i serierekkefølgen, slik at forkastningen får plotting-prioritet. Dersom en serie blir definert som en forkastning skifter GemPy på skalarfeltet slik at forkastningen påvirker modellen.

En diskordans er en grense mellom to deler av en geologisk lagrekke[41]. En geologisk lagrekke er en gruppe som består av flere geologiske lag. I GemPy blir diskordanser konstruert ved å lage to forskjellige serier der de geologiske lagene i seriene overlapper hverandre, som vist i figur 26.



(a) Flate 1 og 2 ligger først i serierekkefølger

(b) Flate 1 ligger først i serierekkefølgen

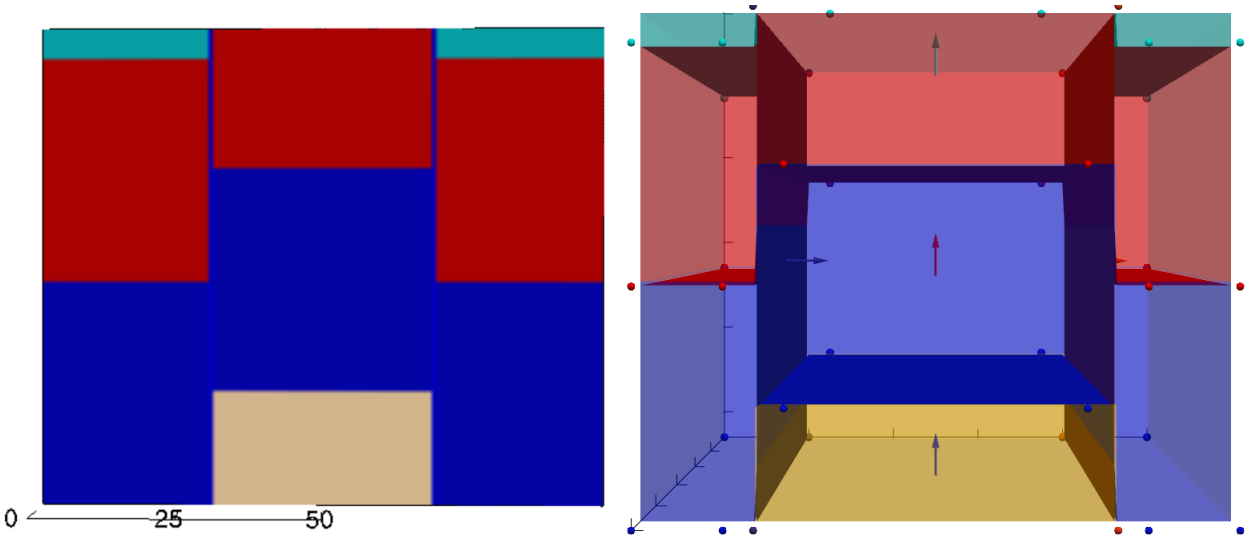
Figur 26: Endring av serierekkefølge

Kompilator

GemPy bruker Theano til å kompilere og visualisere modeller. Theano er et Python-bibliotek som brukes for å optimalisere Python-kompilatoren. I GemPy er hovedfunksjonen til Theano å manipulere og evaluere matematiske utrykk ved hjelp av NumPy[42]. Før en modell kan bli konstruert i GemPy må Theano initialiseres. Theano tar størrelsen, oppløsningen og opasiteten til modellen som skal konstrueres og initialiserer modellen slik at den kan plottes i PyVista og Matplotlib. Størrelsen på modellen er hvor langt x-, y- og z-aksene skal strekkes. Oppløsningen til modellen brukes i PyVista.

En PyVista-modell blir bygget opp av mange små kuber som består av to pyramider. Oppløsningen på modellen definerer hvor mangen slike kuber som skal brukes i x-, y- og z-retning. Opasitet er et mål på ugjennomsiktighet og definerer hvor transparent modellen skal vises i PyVista.

4.3.2 Visible Geology



(a) Visible Geology (b) GemPy

Figur 27: Samme modell konstruert i Visible Geology og GemPy

Visible Geology er et interaktivt og nettbasert modelleringsverktøy for geologisk modellering[43]. Modelleringen i Visible Geology er veldig lik modelleringen i GemPy. Siden oppdragsgiver ønsket et lignende produkt, ble det brukt som inspirasjon til modellerings-modulen. Følgende modelleringsfunksjoner fra Visible Geology er blitt konstruert i GemPy-modelleringsmodulen.

- Konstruering av geologiske lag
- Vipping og rotasjon av geologiske lag
- Forkastning
- Diskordans

4.3.3 Oppbygging av modul

Koden for modulen er delt inn i tre Python-filer og en CSS-fil.

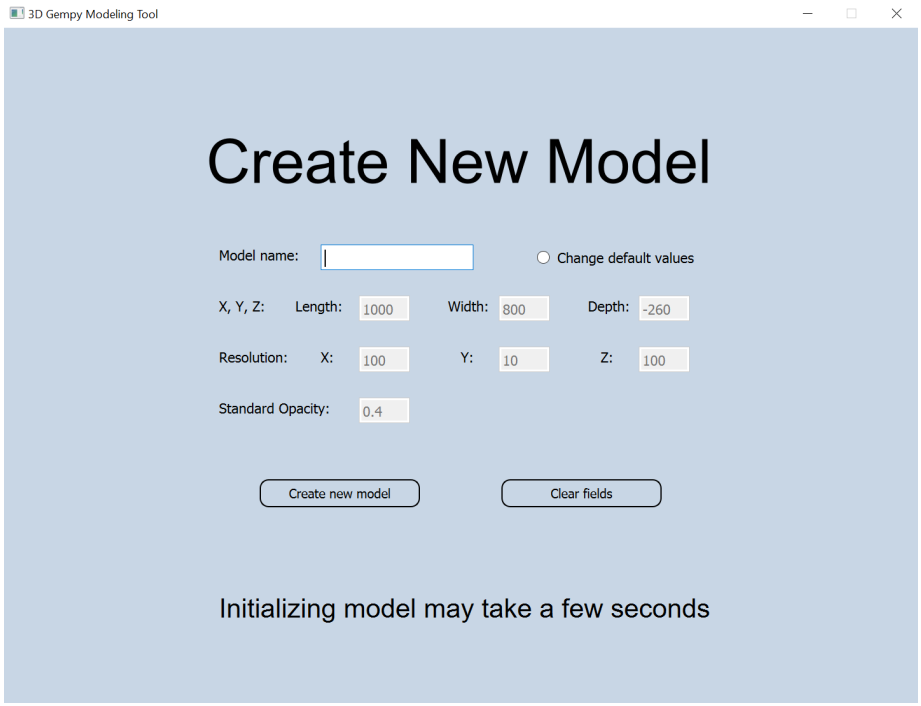
- main.py - Denne filen innenholder klassen "GemPyModelling".
- gempyGUI.py - Denne filen innenholder koden for brukergrensesnittet.
- calculations.py - Denne filen innenholder alle matematiske funksjoner som blir brukt av "GemPyModelling".
- gempyStyle.css - Denne filen innenholder CSS-en til brukergrensesnittet.

Alle modelleringsfunksjonene i "main" tar inn variabler fra "gempyGUI". For hver funksjon som kalles på fra "GemPyGUI" til "main", er det blitt laget en egen funksjon i "gempyGUI". Disse funksjonene validerer alle variablene som blir tatt inn i funksjonen, og prøver å fange unntak. Dersom et unntak fanges, avbrytes funksjonen i "main" og et vindu åpnes i brukergrensesnittet som opplyser brukeren hvorfor feilen oppsto.

```
def add_simple_layer(self, name, height, color):
    try:
        self.g.add_simple_bed(name, float(height), color)
    except NameError:
        self.show_error('Surface already exist. Try a different name.')
    except ValueError:
        self.show_error('Same height is already defined to a different bed.
                                                ')
    except UserWarning:
        self.show_error('Height must be within z coordinate range.')
```

Kode 9: Funksjon i GemPyGUI.py som kaller på funksjon i main.py

4.3.4 Oppstart av modul



Figur 28: Oppstartsvindu

Ved oppstart av modulen vises oppstartsvinduet i figur 28. Oppstartsvinduet tar inn alle variablene som trengs for å initialisere GemPy-modelleringsklassen og for å konstruere en ny modell. Disse variablene er forklart i 4.3.1 under paragrafet "Kompilator". Som standardverdier blir de fysiske målene på sandkassen brukt som størrelse på modellen, mens standard GemPy-verdier blir brukt som oppløsning og opasitet. Disse verdiene kan endres på i brukergrensesnittet dersom det er ønskelig.

Ved å klikke på "Create new model" blir alle verdiene sendt til Python-klassen. Modulen initialiserer GemPy og starter Theano. Det legges så til en serie for kjellerflaten, som legges til i modellen. Til slutt åpnes modellen i 2D og 3D.

```
def create_model(self, name, x_res=100, y_res=10, z_res=100, new_model=True
                                                                        ):
    self.geo_model = gp.create_model(name)
    self.geo_model = gp.init_data(self.geo_model,
                            extent=[0, self.x, 0, self.y, self.z, 0],
                            resolution=[x_res, y_res, z_res])
    gp.set_interpolator(self.geo_model, theano_optimizer='fast_compile',
                                        verbose=[])
    if new_model:
        self.geo_model.add_series('Basement')
        self.geo_model.add_surfaces('basement')
        gp.map_series_to_surfaces(self.geo_model, {'Basement': 'basement'})
        self.plot_2d()
        self.plot_3d()
```

Kode 10: Konstruering av modell i main.py

Når grunnlaget for modellen er konstruert fjernes oppstartsvinduet og et vindu med modelleringsfunksjoner åpnes (Se figur 33).

4.3.5 Modelleringsfunksjoner

GemPy-modelleringsmodulen består av følgende geologiske modelleringsfunksjoner:

- Konstruering av geologiske lag
- Diskordans
- Rotasjon og vipping av geologiske lag
- Forkastning
- Avansert konstruering av geologiske lag
- Avansert forkastning

Alle modelleringsfunksjonene lagrer hvordan modellen ser ut før funksjonen blir kjørt. Denne egenskapen er blitt lagt til for at det skal være mulig å gå tilbake til hvordan modellene så ut før funksjonen ble kjørt, ved bruk av en angreknapp.

Konstruering av geologiske lag
Funksjonen for å konstruere geologiske lag konstruerer et flatt geologisk lag i modellen og tar inn fire argumenter. Disse er flatenavn, starthøyde, farge og serie. Funksjonen sjekker først om det allerede finnes en flate med de samme koordinatene og om z-koordinatene er innenfor modellens høyde. Dersom en av disse tilfellene oppstår, stopper funksjonen og en feilmelding åpnes i brukergrensesnittet. Deretter blir flaten lagt til i GemPy.

Flaten blir så tildelt fire punkter og en orientering. Punktene blir plassert i hvert hjørne av koordinatsystemet med starthøyde-argumentet som z-koordinat, mens orienteringen blir plassert på midtpunktet av flaten og peker vinkelrett opp. En valgt farge blir så tildelt flaten. Dersom argumentet står tomt vil flaten få en tilfeldig farge. Fargen blir så lagret i en oppslagsliste hvor flatenavnet er nøkkelen.

```
def compute_and_plot(self):
    gp.compute_model(self.geo_model)
    for index in range(len(self.geo_model.surfaces.df.surface)):
        if self.geo_model.surfaces.df.surface[index] not in
                                                self.custom_color_dict:
            self.custom_color_dict[self.geo_model.surfaces.df.
                surface[index]] = self.geo_model.surfaces.df.color[index]
    self.geo_model.surfaces.colors.change_colors(self.custom_color_dict)
    self.p2d.remove(self.ax)
    self.p2d.plot_data(self.ax, cell_number=1)
    self.p2d.plot_lith(self.ax, cell_number=1)
    self.p2d.plot_contacts(self.ax, cell_number=1)
    self.p3d.plot_surface_points()
    self.p3d.plot_data()
    self.p3d.plot_surfaces()
    self.p3d.plot_structured_grid(opacity=self.opacity)
```

Kode 11: compute_and_plot() kjøres hver gang modellen endres på

Metoden i kode 11 kjører hver gang modellen endres på. Metoden beregne modellen med interpolasjonsalgoritmen til GemPy, lagrer fargen på flatene i en oppslagsliste og plotter modellen.

Dersom man ønsker å fjerne flaten er det mulig å klikke på "Undo layer" i brukergrensesnittet. Programmet henter da den siste flaten som ble konstruert og sletter flaten.

Diskordans

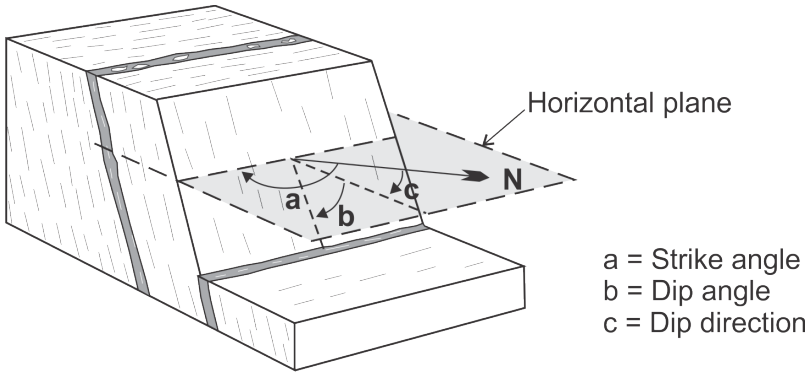
For å konstruere en diskordans i GemPy må en ny serie konstrueres, tildeles flater og legges til i begynnelsen av GemPy serierekkefølgen. Den får da plotting-prioritet over alle andre serier i modellen. Dette kan gjøres manuelt eller ved hjelp av diskordans modelleringsmetoden.

Metoden for å konstruere diskordanser tar inn tre argumenter. Disse er serienavn, dybde og tykkelse. Dybden bestemmer hvilket z-koordinat diskordansen skal starte på. Tykkelsen bestemmer hvor tykk en flate i diskordansen skal være. Funksjonen starter med å legge til en ny serie. Etter serien er lagt til regner funksjonen ut hvor tykk den øverste flaten i diskordansen skal være ved bruk av modulo. Utregningen gjøres ved å ta dybdeargumentet modulo flatetykkelseargumentet. Modulo er en matematisk operasjon som brukes til å finne resten av et heltall etter en divisjon med et annet heltall[44].

Funksjonen starter så å plotte flere geologiske lag fra dybden som ble gitt til metoden med intervaller på tykkelsen som ble gitt til metoden. Til slutt blir serien lagt til i begynnelsen av GemPy-serierekkefølgen. Alle flatene blir konstruert på samme måte som beskrevet i forrige paragraf om konstruering av geologiske lag.

Modulen har også en angremetode for diskordanser. Metoden tar den siste diskordansen som ble konstruert og sletter serien sammen med all tilhørende data.

Rotasjon og vipping av geologiske lag



Figur 29: Strøk og fall(Strike and dip)[45]

For geologiske flater brukes begrepene strøk og fall for å beskrive hvordan flaten er orientert i rommet. Strøk beskriver retningen på skjæringslinjen mellom flaten og horisontalplanet i forhold til nord, mens fall beskriver vinkelen mellom flaten og horisontalplanet (se figur 29) [46].

Metoden for rotasjon og vipping tar inn disse to begrepene som argumenter sammen med serien som skal endres på. Metoden itererer gjennom hver eneste flate i serien og lagrer koordinatpunkter, orienteringspunkt og orienteringsvinkel til flatene i en oppslagsliste. Dette gjøres i tilfelle en bruker ønsker å angre endringene på modellen. Etter dette blir nye flatepunkter generert ved bruk av funksjonen i kode 12.

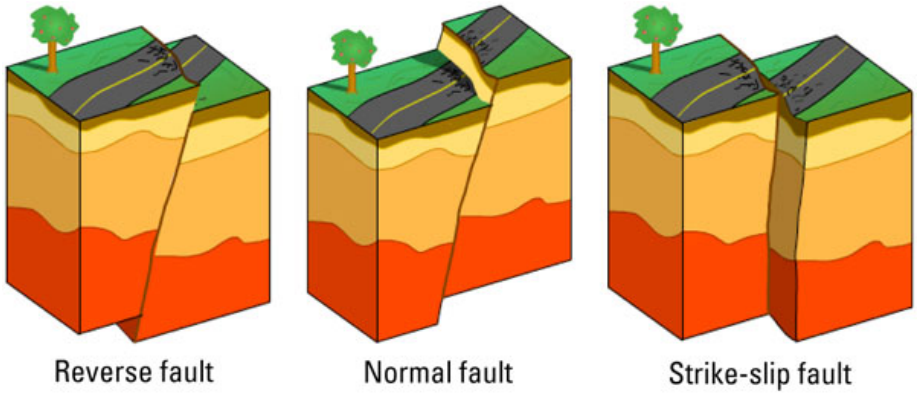
Funksjonen tar inn punktene til flaten som skal roteres, strøk, fall, hvilket punkt flaten skal orienteres rundt og størrelsen på koordinatsystemt i x-, y- og z-retning. Ved bruk av strøk- og fall-argumentene konstrueres en rotasjonsmatrise som brukes for å rotere alle punktene til flaten. Funksjonen fit_points() kjøres så for å flytte på punktene slik at de er innenfor koordinatsystemet. Nye orienteringer blir så plassert

i midten av hver flate med strøk og fall som retningsvektor. Til slutt blir den nye orienteringen lagret slik at det er mulig å rotere flatene flere ganger om ønskelig.

```
def rotate_point(points, strike, dip, orientation_point, x_border, y_border
                                        , z_border):
    point_list = []
    cosa = math.cos(-strike)
    sina = math.sin(-strike)
    cosb = math.cos(dip)
    sinb = math.sin(dip)
    cosc = math.cos(math.radians(0))
    sinc = math.sin(math.radians(0))
    rotation_matrix = np.array((
        (cosa * cosb, cosa * sinb * sinc - sina * cosc, cosa * sinb * cosc
                                                + sina * sinc),
        (sina * cosb, sina * sinb * sinc + cosa * cosc, sina * sinb * cosc
                                                - cosa * sinc),
        (-sinb, cosb * sinc, cosb * cosc)))
    op_vector = np.array([orientation_point[0], orientation_point[1],
                                            orientation_point[2]])
    for point in points:
        point_vector = np.array([point[0], point[1], point[2]])
        new_points = rotation_matrix * (point_vector - op_vector)
        x = new_points[0][0] + new_points[0][1] + new_points[0][2] +
                                                op_vector[0]
        y = new_points[1][0] + new_points[1][1] + new_points[1][2] +
                                                op_vector[1]
        z = new_points[2][0] + new_points[2][1] + new_points[2][2] +
                                                op_vector[2]
        point_list.append([x, y, z])
    fit_points_list = self.fit_points(point_list, strike, x_border,
                                            y_border, z_border)
    return fit_points_list
```

Kode 12: Funksjon for å rotere og vippe flater

Forkastning



Figur 30: Reversforkastning, Normalforkastning og Sidelengsforkastning[47]

For å konstruere en forkastning i GemPy må koordinatpunktene til en flate ha forskjellige z-koordinater på begge sidene av forkastningflaten. Metoden for forkastning tar inn 7 argumenter. Argumentene er flatenavn, serienavn, startkoordinat til flaten, strøk, fall, "rake" og "slip".

"Rake" er vinkelen mellom en flate og strøklinjen til flaten. Den definerer hvilken retning flatene skal gli i forhold til hverandre (se figur 30)[48]. "Slip" er hvor langt flaten glir under forkastningen.

Metoden starter med å definere en ny forkastningsserie og legger denne i begynnelsen av GemPy serierekkefølgen. En ny forkastningsflate blir så konstruert ved bruk av kode 12 hvor koordinat-argumentet i metoden blir brukt som orientering-punkt i funksjonen. Flaten blir så tildelt en orientering på midtpunktet av flaten. Metoden bruker deretter ligningen til et plan for å finne ut hvilken punkter som ligger bak og foran forkastningsflaten.

```
# Thanks to Nestor Cardozo for this function
def slip_coords(strike, dip, slip, rake):
    strike_s = slip * math.cos(rake)
    dip_s = slip * math.sin(rake)
    east_s = math.sin(strike) * strike_s + math.cos(strike) *
                                           math.cos(dip) * dip_s
    north_s = math.cos(strike) * strike_s - math.sin(strike) *
                                           math.cos(dip) * dip_s
    up_s = -math.sin(dip) * dip_s
    return [east_s, north_s, up_s]
```

Kode 13: Funksjon for endring av punkter foran forkastningflaten

Funksjonen i kode 13 tar så inn argumentene strøk, fall, "rake" og "slip" og returnerer en retningsvektor. Retningsvektoren blir addert til punktene foran forkastningsflaten som fører til at punktene endrer posisjon i koordinatsystemet.

For at modulen skal ha støtte for flere forkastninger i samme modell må hver flate som er påvirket av forkastningene ha minimum et koordinatpunkt mellom hver forkastning. Følgende funksjoner blir brukt for å finne hvor disse punktene skal plasseres.

Funksjonen i kode 14 tar inn to lister som argumenter. En liste med punkter bak forkastningsflaten, og en liste med punkter foran. Funksjonen blir brukt for hver flate som påvirkes av forkastningen. Disse listene blir så lagt til et k-d tre for å finne ut hvilke to punkter bak forkastningsflaten som ligger nærmest to punkter foran forkastningsflaten. To punkter bak flaten blir paret opp med to punkter foran flaten og blir returnert som en liste. Metoden lager deretter to ligninger for rette linjer ut ifra disse punktene.

Funksjonen i kode 15 tar deretter inn argumenter for normalen til forkastning-flaten, et punkt på forkastning-flaten, ligningen for en rett linje og et punkt på denne linjen. Dette gjøres for begge linjene. Funksjonen returnerer hvor i koordinatsystemet den rette linjen krysser forkastningsplanet. Ut ifra dette koordinatpunktet blir et nytt punkt konstruert på hver side av forkastningflaten til hver flate påvirket av forkastningen.

```
def nearest_point(point_list_1, point_list_2):
    if len(point_list_1) <= 1:
        point_list_1 = [point_list_1[0], point_list_1[0]]
    if len(point_list_2) <= 1:
        point_list_2 = [point_list_2[0], point_list_2[0]]
    point_pair_1 = [point_list_1[spatial.KDTree(point_list_1).
                    query(point_list_2)[1][0]],
                    point_list_2[spatial.KDTree(point_list_2).
                    query(point_list_1)[1][0]]]
    point_pair_2 = [point_list_1[spatial.KDTree(point_list_1).
                    query(point_list_2)[1][1]],
                    point_list_2[spatial.KDTree(point_list_2).
                    query(point_list_1)[1][1]]]
    closest_point = [point_pair_1, point_pair_2]
    return closest_point
```

Kode 14: Funksjon for å finne to par med nærmest punkter

```
# gist.github.com/TimSC/8c25ca941d614bf48ebba6b473747d72
def LinePlaneCollision(planeNormal, planePoint, rayDirection, rayPoint,
                                        epsilon=1e-6):
    ndotu = planeNormal.dot(rayDirection)
    if abs(ndotu) < epsilon:
        raise RuntimeError("no intersection or line is within plane")
    w = rayPoint - planePoint
    si = -planeNormal.dot(w) / ndotu
    Psi = w + si * rayDirection + planePoint
    return Psi
```

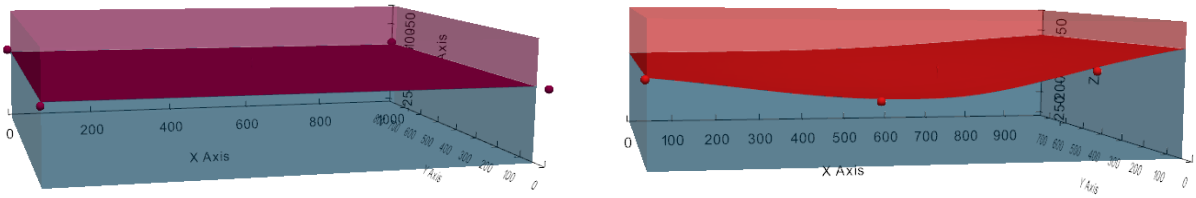
[49]

Kode 15: Funksjon for å finne hvor en linje krysser et plan

De gamle punktene til flatene påvirket av forkastningen blir lagret i en oppslagsliste dersom brukeren ønsker å angre forkastningen. Dersom brukeren angrer på forkastningen blir serien til forkastningen slettet. For å finne indeksen på punktene som ble lagt til ved forkastningen, blir lengden på listen av gamle punkter sammenlignet med lengden på listen av nåværende punkter. Disse punktene fjernes så fra modellen. Dette gjøres for hver eneste flate påvirket av forkastningen. Til slutt blir de gjenværende punktene endret til verdiene av de gamle punktene.

Avansert konstruering av geologiske lag
Ved avansert konstruering av geologiske lag kan brukeren konstruere flater med egendefinerte koordinatpunkter og orienteringer. Dette fører til at det er mulig å lage mer avanserte flater og utnytte GemPy sitt fulle potensiale.

Metoden for avansert konstruering av geologiske lag tar inn alle koordinatpunktene som er gitt av en bruker i brukergrensesnittet og plotter punktene i koordinatsystemet til modellen. Deretter kompilerer GemPy-modellen før flatene blir plottet i koordinatsystemet.



(a) Vanlig konstruering

(b) Avansert konstruering

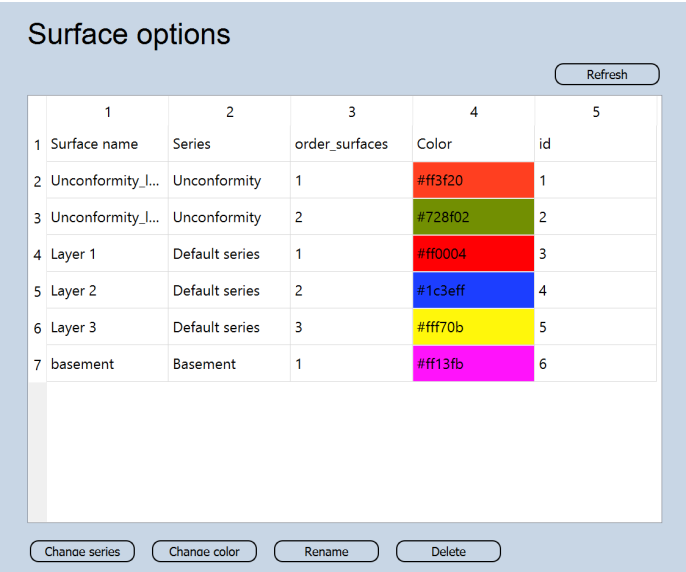
Figur 31: Vanlig og avansert konstruering av flater

Avansert forkastning
Avansert forkastning fungerer helt likt som avansert konstruering av geologiske lag. Den eneste forskjellen er at serien blir definert som en forkastningsserie i GemPy.

4.3.6 Datastrukturer og datamanipulasjon

GemPy bruker Python-biblioteket "Pandas" for håndering og manipulasjon av data. "Pandas" tilbyr lagring og manipulasjon av data i numeriske tabeller[50]. GemPy har en slik tabell for serier, flater, koordinatpunkter, orienteringer, forkastninger og tilleggsdata.

Modelleringsmodulen har metoder for manipulasjon av serie- og flate-data. Denne dataen vises også i brukergrensesnittet under "Series options" og "Surface options".



Figur 32: Flateinnstillinger

Dataen hentes fra Pandas-tabellen og legges til en liste. Listen sendes så videre til brukergrensesnittklassen hvor den blir vist i en tabell (se figur 32).

```
def surfaces_to_data_list(self):
    self.surfaces_data = []
    if len(self.surfaces_to_series) > 1:
        for index in range(len(self.geo_model.surfaces.df.index)):
            self.surfaces_data.append(
                [self.geo_model.surfaces.df.surface[index],
                self.geo_model.surfaces.df.series[index],
                str(self.geo_model.surfaces.df.order_surfaces[index]),
                self.geo_model.surfaces.df.color[index],
                str(self.geo_model.surfaces.df.id[index])])
        list.sort(self.surfaces_data, key=lambda x: int(x[4]), reverse=
                                                    False)
    return self.surfaces_data
```

Kode 16: Funksjon for å hente data fra en Pandas-tabell

Manipulasjon av seriedata

Modulen innenholder fire valg for manipulasjon av seriedata:

- Lag en ny serie
- Endre serierekkefølge
- Endre serienavn

- Slett serie

Dersom en bruker velger å lage en ny serie blir en inaktiv serie konstruert. Serien er inaktiv helt til brukeren konstruerer en flate og legger denne flaten til serien. Endring av serierekkefølge endrer plottings-prioriteten til serien avhengig om den blir plassert tidlig eller sent i serierekkefølgen. Endring av serienavn endrer navn på serien og oppdaterer alle datastrukturer hvor serienavnet er lagret, med det nye navnet på serien. Sletting av en serie sletter selve serien sammen med all tilhørende data. Serien blir også slettet fra alle datastrukturer som innenholder serienavnet.

Manipulasjon av flatedata
Modulen innenholder fire valg for manipulasjon av flatedata:

- Endre serie
- Endre farge
- Endre flatenavn
- Slett flate

Ved endring av en serie fjernes flaten fra dens nåværende serie og legges til en ny serie spesifisert av brukeren. Endring av farge endrer fargen på flaten og oppdaterer farge-oppslagslisten med den nye fargen. Endring av flatenavn endrer navnet på flaten og oppdaterer alle datastrukturer hvor flatenavnet er lagret, med det nye navnet på flaten. Sletting av en flate sletter selve flaten og alt av data flaten innenholder (koordinatpunkt og orientering). Etter at flaten er slettet nullstilles indeksene til Pandas-tabellen for flater og koordinatpunkt.

Lagre modell
GemPy kan lagre modeller i to type filer. Disse er CSV og "Pickle". I modulen ble Pickle brukt som filformat. Pickle blir brukt for å serialisere et Python-objekt til en binærfil eller til å parse en binærfil til et Python-objekt[51].

Metoden for lagring tar inn et filnavn og en sti oppgitt av bruker i brukergrense-snittet og lagrer modellen som en pickle-fil.

```
def save_model(self, name, path):
    gp.save_model_to_pickle(self.geo_model, path=path + "/" + name)
```

Kode 17: Metode for å lagre en GemPy-modell

Laste inn modell
Modulen kan laste inn GemPy-modeller i filformatet "Pickle". Metoden for lasting av modeller tar inn tre argumenter. Disse er filnavn, sti og en boolsk variabel som avgjører om metoden skal plotte modellen i Matplotlib og PyVista.
Modelleringsklassen innenholder fire datastrukturer for intern lagring av variabler:

- surface_points - En oppslagsliste som innenholder flater og koordinatpunktene til flatene.
- custom_color_dict - En oppslagsliste som innenholder fargene til de forskjellige flatene i modellen.
- surface_to_series - En oppslagsliste som innenholder flater og hvilken serie flatene ligger i.
- series_order - En liste som innenholder rekkefølgen til seriene i modellen.

Metoden for lasting av modeller går gjennom Pandas-tabellene til GemPy-modellen og fyller ut disse fire datastrukturene slik at modellen kan endres på etter den er lastet inn i modulen.

```
def file_load(self):
    name = Qtw.QFileDialog.getOpenFileName(self, 'Open Model')
    path, filename = os.path.split(name[0])
    project_name = str(filename).split("_")[0]
    self.g.load_model(project_name, path, plot=False)
    model_information = self.g.get_model_information()
    self.g = GemPyModelling(x=int(model_information[0]),
                            y=int(model_information[1]),
                            z=int(model_information[2]),
                            opacity=float(0.4))
    self.g.create_model(filename,x_res=int(model_information[3],
                                y_res=int(model_information[4]),
                                z_res=int(model_information[5]),
                                plot=False)
    self.g.load_model(project_name, path, plot=True)
```

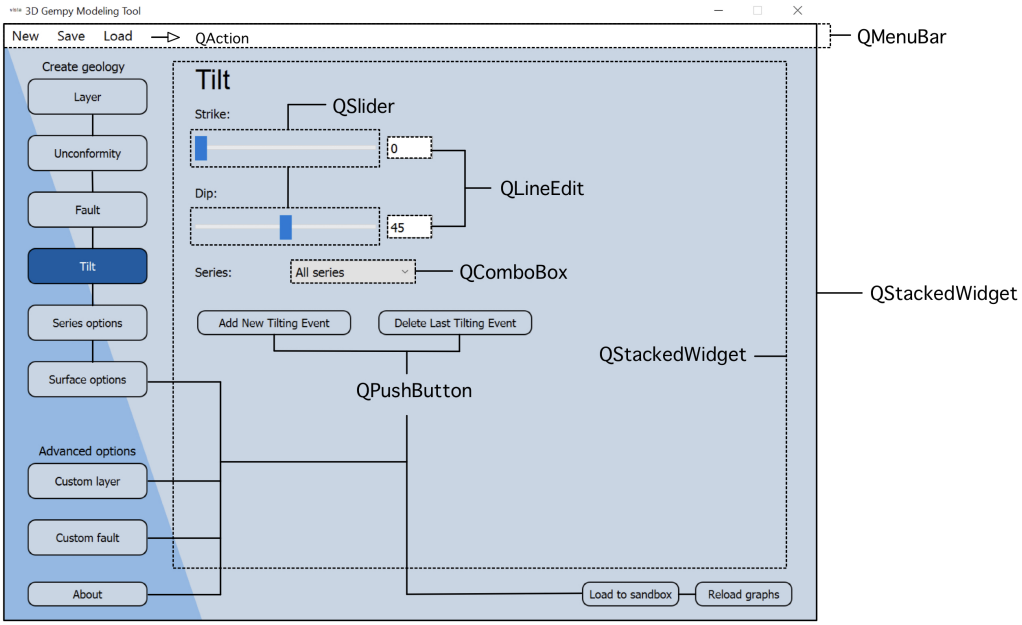
Kode 18: Metode i brukergrensesnitt klassen som laster inn en modell

Metoden kalles på to ganger fra brukergrensesnittklassen (se kode 18). Den første gangen metoden kalles på blir ikke modellen plottet og klassens datastrukturer forblir uendret. Dette gjøres siden modulen trenger informasjon om modellen (størrelse og oppløsning). En ny instans av klassen blir så konstruert og en ny modell blir initialisert med informasjonen som er blitt hentet ut av pickle-filen. Metoden kalles på en gang til, men denne gangen blir modellen plottet og datastrukturene fylles ut.

Projisere modell til sandkassen
For å projisere en modell i sandkassen må en pickle-fil bli konstruert ut ifra modellen før den blir lastet inn i en Jupyter Notebook-fil og projisert til sandkassen via GemPy-modulen. Filen blir lagret i GemPy-modulmappen til Open AR Sandbox. Dette gjøres på samme måte som i kode 17.

Mer om hvordan modellen blir projisert i sandkassen kan leses i kapittel 4.2.6.

4.3.7 Brukergrensesnitt



Figur 33: Oppbygging av brukergrensesnitt

Brukergrensesnittet er delt i tre deler. Disse er QMainWindow, PyVista og Matplotlib.

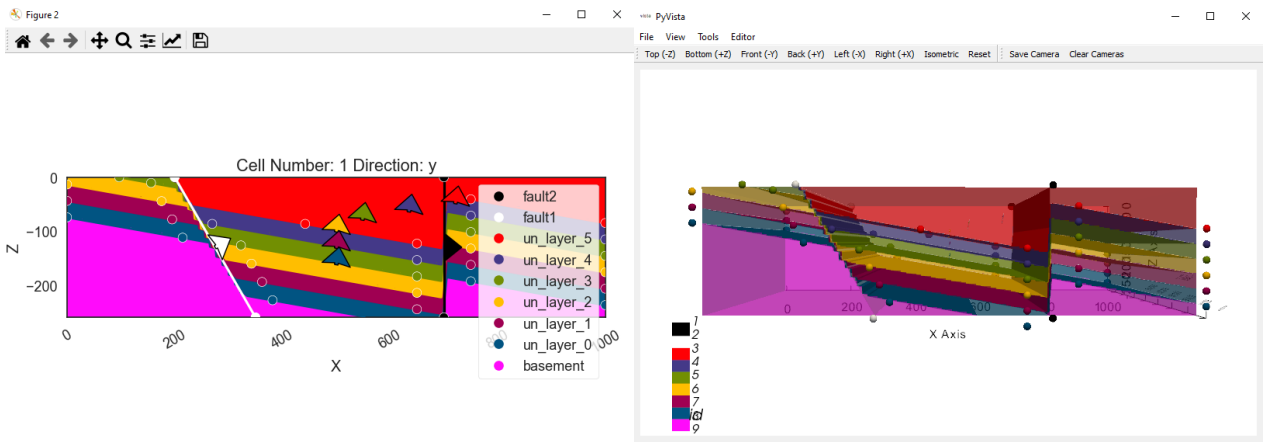
QMainWindow
QMainWindow består av to QStackedWidgets, en ytre og en indre:

1. Den ytre QStackedWidgeten (i figur 33) består av to QWidgeter:
    - Den ene QWidgeten brukes til å initialisere modellen, som vises i figur 28.
    - Den andre QWidgeten består av de ulike modelleringsfunksjonene til programmet (hele vinduet i figur 33).

2. Den indre QStackedWidgeten brukes for å vise innholdet til de ulike menyvalgene i brukergrensesnittet.

Etter modellen er initialisert, vises About-widgeten. Den innenholder informasjon om hvordan en bruker skal konstruere en modell og limitasjoner til GemPy. Brukergrensesnittet består også av en menylinje som bruker PyQT-klassen QMenuBar. Denne menylinjen innenholder funksjonene for lagring av modell, lasting av modell og konstruering av en ny modell.

Matplotlib og Pyvista



(a) Matplotlib (b) Pyvista

Figur 34: Modell i Matplotlib og Pyvista

Når en modell skal konstrueres åpnes et Matplotlib og Pyvista vindu ved siden av QMainWindow. Disse viduene oppdateres automatisk dersom det skjer noen endringer i modellen og blir brukt for å visualisere modellen i 2D og 3D. Modellen i figur 34 blir projisert i sandkassen i figur 25.

Validering av inndata

For å validere inndata fra brukergrensesnittet brukes PyQT-klassen QValidator. QValidator validerer inndatafelt (QLineEdit) noe som gjør at inndatafeltet kun godkjenner forhåndsbestemte verdier. QValidator er delt inn i tre underklasser for forskjellig type validering. QIntValidator blir brukt til validering av heltall. QDoubleValidator blir brukt til validering av flyttall. Dette gjøres ved å definere et tallområde med godkjente verdier. QIntValidator blir brukt for å validere at tall som skal sendes

fra brukergrensesnittet til modelleringsmodulen er innenfor en en viss tallgrense.

```
    depth_in.setValidator(Qtg.QIntValidator(self.g.get_z(), 0))
```

Kode 19: Kodelinje som setter godkjent inndataverdi mellom den laveste godkjente z-verdien og 0

QRegExpValidator gir egendefinert kontroll for validering ved hjelp av regulære utrykk. Et regulært utrykk er en streng som beskriver et mønster av strenger. I QRegExpValidator er dette mønsteret regler på hvordan andre strenger kan skrives.

```
    self.hex_regex = Qtc.QRegExp("^#([0-9A-Fa-f]{6})$")
```

Kode 20: Regulært utrykk for heksadesimale fargekoder

Det regulære utrykket i kode 20 definerer at strengen skal starte med en emneknagg etterfulgt av seks symboler som skal være en kombinasjon av tall mellom 0 og 9 og bokstaver fra A til F. Utrykket blir brukt for å validere egendefinerte farger av flater.

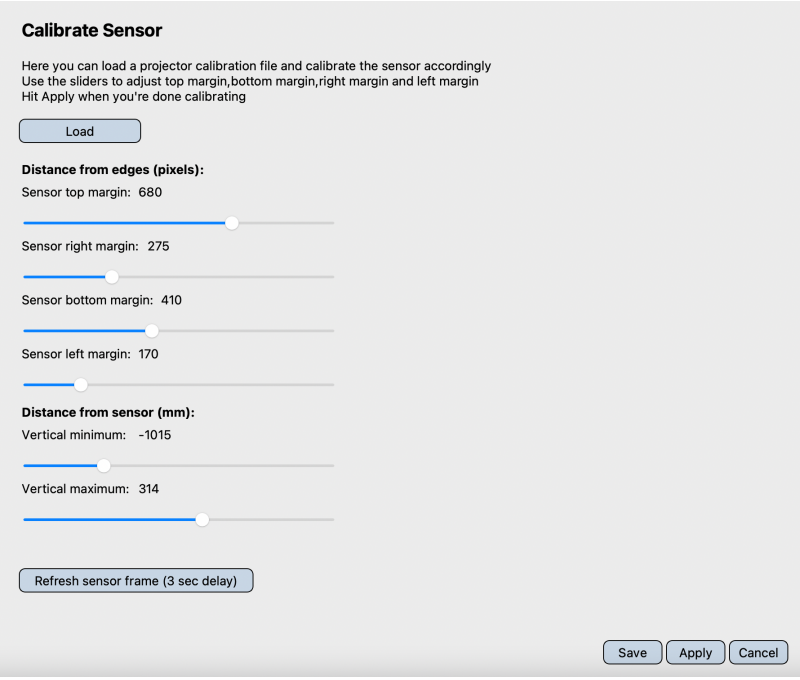
5 Diskusjon

Dette kapittelet tar for seg begrensninger som oppsto underveis, muligheter for videreutvikling og hva som skal til for at programmet skal fungere optimalt.

5.1 Førsteutkast av brukergrensesnittet

Til å begynne med ble utviklet et brukergrensesnitt som var tiltenkt å kunne kjøre frittstående, uten bruk av Jupyter-filene til Open AR Sandbox. Dette viste seg å være en tidkrevende tilnærming til oppgaven ettersom hvert element i brukergrensesnittet måtte kobles direkte mot programvaren til Open AR Sandbox. For få til dette måtte hele programvaren til Open AR Sandbox rekonstrueres for å passe elementene i brukergrensesnittet.

Dette ble løst ved å heller modifisere Jupyter Notebook-filene, som nevnt i delkapittel 4.1.1. Videre ble Jupyter Notebook-funksjonaliteten implementert direkte i det nye brukergrensesnittet ved bruk av QWebEngineView.

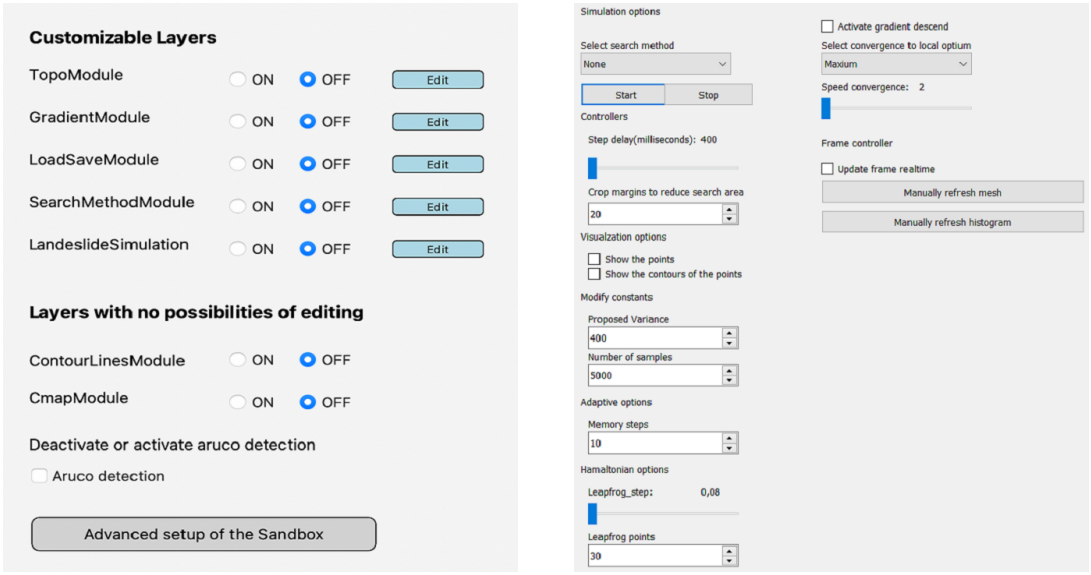


Figur 35: Førsteutkast av sensorkalibreringsvindu

I delmenyen "Calibrate Sensor" som vist på figur 35 var det lagt til rette for å kalibrere sensoren med skyvebrytere, samt lagre kalibreringsfilen og laste inn gamle filer. For kalibrering av projektor ble det laget et tilnærmet likt vindu. For en lettere kalibrering av sensor- og projektor var det tiltenkt et lite vindu som skulle vise endringer i sanntid, ved siden av skyvebryterne.

5.1.1 Aktivering av moduler

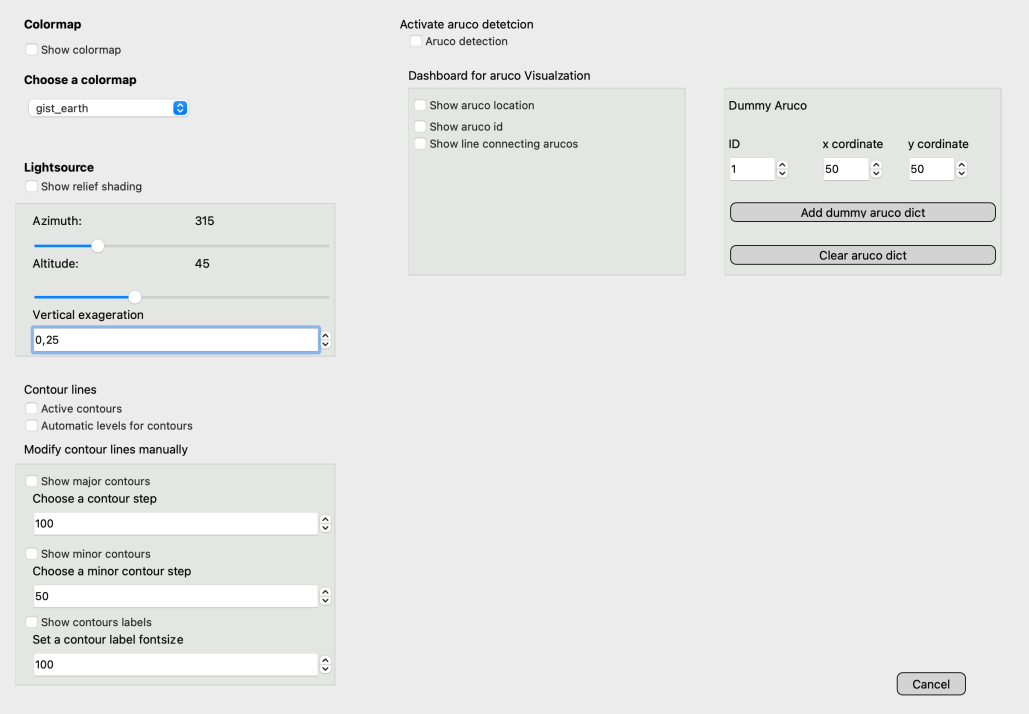
I delmenyen "Modules" var det tilrettelagt for å kunne velge ut hvilke moduler som skulle aktiveres. Som standard var alle modulene skrudd av, som vist i figur 36a. Hver modul var utstyrt med en egen edit-knapp. Ved bruk av en edit-knapp ble det åpnet et vindu hvor det skulle være mulig å endre på innstillengen til modulene. Idéen var at Open AR Sandbox kunne kjøre samtidig som en bruker kunne velge hvilke moduler som skulle være aktive. I figur 36b vises modifiseringsvinduet til "Search Method Module".



(a) Modulside (b) Search Method Module-innstilinger

Figur 36: Endring av modulinnstillinger

5.1.2 Beslutning om å ikke gå videre med førsteutkast



Figur 37: Tiltenkt kontrollpanel til server

Under konstruksjon av applikasjonens førsteutkast ble det hentet inspirasjon fra de originale Jupyter Notebook-filene til Open AR Sandbox. Etter nøye vurdering ble det besluttet å ikke gå videre med ideen. Alt arbeidet som ble lagt inn under konstruksjonen av førsteutkastet førte til en større og bredere kompetanse om Open AR Sandbox samt PyQt. Dette gjorde at omstillingen til et nytt design, gikk raskt og effektivt på grunn av kompetansen som ble opparbeidet under utvikling av førsteutkastet.

5.2 Bruken av BAT

Det har blitt laget en BAT-fil for starting av applikasjonen fra skrivebordet. BAT-filen består av terminalkommandoer som kjøres sekvensielt når den blir åpnet. BAT-filer kan redigeres ved bruk av en teksteditor[52].

For å åpne brukergrensesnittet til Open AR Sandbox brukes snarveien på skrivebordet. Fordelen med å bruke en BAT-fil til å starte applikasjonen er at det går raskere enn å starte den manuelt gjennom terminalvinduet. Ulempen med å bruke

en BAT-fil til oppstart er at hele filstien er spesifisert. Dersom en bruker endrer lokasjonen på noen av filene, vil BAT-filen ikke lenger være brukelig. Hvis en bruker endrer på lokasjonen, må filstien i BAT-filen korrigeres.

```
echo %cd%
call C:\Users\sandy\anaconda3/Scripts/activate.bat C:\Users\sandy\anaconda3
call conda activate app-env
cd "C:\Users\sandy\Desktop\SEB_Addon\mainGUI"
python sandbox.py
```

Kode 21: Koden til BAT-filen, som er ikonet på skrivebordet

Oppstartsfilen til brukergrensesnittet er vist i kode 21. BAT-filen starter Anaconda-miljøet for prosjektet. Filen navigerer så til filstien hvor brukergrensesnittet er lagret og åpner brukergrensesnittet ved hjelp av Python-kommandoen.

5.3 Forbedring av maskinvare

Det kreves mye prosessorkraft for å kjøre Open AR Sandbox optimalt. For at geologistudenter ved UiS skal få utnyttet programmets fulle potensiale, er det behov for å oppgradere den nåværende datamaskinen. Ved flere tilfeller har prosessorbruken til datamaskinen ligget på 80-90 prosent ved kjøring av programmet. Dette har gjort at datamaskinen ikke har fått nok tid til å plotte alle objektene i sandkassen og har forårsaket flimring i det projiserte bildet. Plottingsintervallet til Open AR Sandbox har derfor blitt endret fra standardverdien 0,01 til 0,025 millisekunder. Dette har redusert prosessorbruken betydelig, på bekostning av en lavere oppdateringsrate.

Siden Python ikke er optimalisert for bruk av flere kjerner eller tråder anbefales det å kjøpe en mikroprosessor med høy klokkefrekvens. En klokkefrekvens er antall ganger en mikroprosessor kan utføre en handling i løpet av et sekund. En høyere klokkefrekvens vil føre til at datamaskinen bruker mindre tid på å plotte objekter i sandkassen.

Ettersom applikasjonen krever mye skjermplass, anbefales det å kjøpe en ekstra monitor. Med en ekstra monitor kan det nye brukergrensesnittet til Open AR Sandbox vises på en monitor, samtidig som GemPy-modelleringsmodulen vises på den andre.

5.4 Linux og Microsoft Kinect v1

I de første ukene av prosjektet ble det brukt en Microsoft Kinect v1-sensor. Ved å bruke Kinect v1-sensoren var det kun mulig å kjøre Open AR Sandbox sammen med operativsystemet Linux. Ulempen med Linux-versjonen av Open AR Sandbox er at

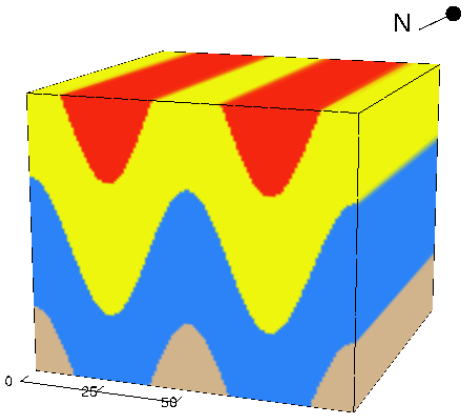
den er ustabil. Dette førte til flikring av det projiserte bildet og at programmet ofte avsluttet med en rekke feilkoder.

Etter samtaler med oppdragsgiver Professor Nestor Fernando Cardozo Diaz ble det besluttet å gå til innkjøp av en Microsoft Kinect v2-sensor. Dette gjorde det mulig å bruke Windows-versjonen av Open AR Sandbox, som også er mer stabil enn Linux-versjonen. Dette har ført til en bedre brukeropplevelse.

5.5 Videreutvikling

I dette delkapitellet vil det bli diskutert funksjoner og egenskaper som gruppen vurderte å implementere i programmet. På grunn av tidsbegrensninger og andre prioriteringer ble ikke disse funksjonene og egenskapene implementert.

5.5.1 Folding i GemPy-modelleringsmodulen



Figur 38: Folding i Visible Geology

Den neste planlagte modelleringsfunksjonen var folding, men på grunn lav prioritering for denne funksjonen av oppgavegiver ble den ikke utviklet. Folding er et geologisk fenomen som oppstår når geologiske flater blir utsatt for høy varme og høyt trykk. Flatene blir bøyelige noe som fører til at de bøyer og folder seg[53].

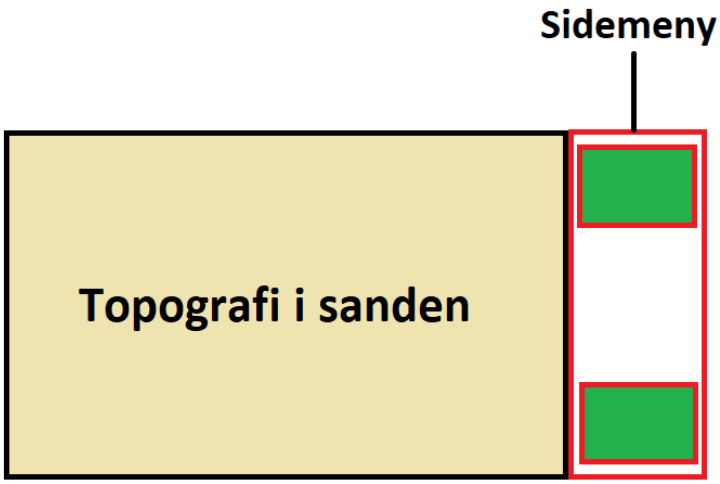
For å lage denne modelleringsfunksjonen trengs en funksjon som plotter punkter for alle flatene som skal påvirkes av foldingen langs en sinuskurve. Hver flate må bli tildelt en sinusfunksjon som starter på flatens z-verdi og nye punkter må plottes langs denne kurven.

Metoden for folding trenger fem argumenter. Disse argumentene er strøk, fall, "rake", periode og amplitude. Perioden og amplituden bestemmer hvordan sinuskurven skal se ut, mens strøk, fall og "rake" bestemmer retningen til sinuskurven. "Rake" er vinkelen mellom horisontalplanet og strøklinjen til planet(se figur 29). Ved å modifisere kode 12 er det mulig å rotere punktene til sinuskurven. Dette gjøres ved å legge til et nytt argument for "rake" i funksjonen og endre på variablene som ligger i kode 22.

```
    cosc = math.cos(rake)
    sinc = math.sin(rake)
```

Kode 22: Nye variabler for å vippe punkter med "rake"

5.5.2 Utbygging av sandkassen med sidemeny



Figur 39: Sandkassen med potensielt informasjons-felt

I Open AR Sandbox er det implementert muligheten til å vise flere informasjonsfelt som kan vise ekstra informasjon om topografien ved siden av sandkassen.

Informasjonsfeltene kan redigeres til å vise relevant informasjon for hver modul. Eksempler kan være:

- Navnene til de ulike geologiske lagene som projiseres i sandkassen
- Informasjon om ArUco-merker
- 2D- og 3D-modeller av GemPy-modulen

For å ta i bruk denne funksjonaliteten må sandkassen som brukes ved UiS bygges ut med en platestruktur til høyre for sandkassen, som vist i figur 39. Videre er det mulig å implementere ny informasjon, ved å bruke kildekoden som er tilgjengelig i originalprosjektet til Open AR Sandbox.

5.5.3 Gjøre om til frittstående applikasjon

Det er mulig å gjøre programmet uavhengig av Jupyter Notebook, slik det var tiltenkt ved starten av prosjektet. Ved å gjøre programmet uavhengig av Jupyter Notebook, vil det ikke være behov for at en Jupyter Notebook-server kjøres i bakgrunnen. For å gjøre dette må blant annet referansene til Bokeh og Panel fjernes fra den originale kildekoden til Open AR Sandbox.

6 Konklusjon

Hovedmålene for oppgaven ble i bunn og grunn oppnådd. Gruppen hadde sett for seg et frittstående brukergrensesnitt, som ikke tok i bruk Jupyter Notebook-filer for å kjøre funksjonaliteten til Open AR Sandbox. Selv om dette ikke ble oppnådd, føler gruppen at brukeropplevelsen i det nye brukergrensesnittet er forbedret.

Det tok lengre tid enn antatt å bli kjent med programvaren til Open AR Sandbox, på grunn av den omfattende mengden med kildekode. Andre ting som ikke gikk helt som planlagt var problemene som oppsto ved bruk av Kinect v1-sensoren. For å ta i bruk v2-sensoren måtte gruppen gå fra å bruke Linux til Windows som operativsystem. Programvaren måtte da settes opp på nytt, noe som ble en tidkrevende prosess.

En gjennomgang av oppsett og modulene er blitt demonstrert i en video som er tilgjengelig på gruppens GitHub-side. En brukermanual for bruk av programvaren er også vedlagt, denne er skrevet på engelsk.

GemPy-modellereringsmodulen har fått de funksjonalitetene som var ønsket av oppdragsgiver. Applikasjonen har blitt testet grundig underveis, og flere bugs er fjernet. Samtidig er det vanskelig å forutse fremtidige bugs som kan oppstå ettersom modulen ikke er blitt brukt over lengre tid.

Dette er første gangen gruppen har utviklet en applikasjon av denne størrelsen, noe som har vært lærerikt. Arbeidet med oppgaven har også gitt gruppen en solid innføring i et helt nytt fagfelt. Gruppen hadde tilnærmet ingen erfaring innen geologi fra før. Det nye fagfeltet har bydd på flere spennende utfordringer og læringsutbyttet er noe gruppen kommer til å ta med seg videre.

Referanseliste

Referanser

[1] "Open ar sandbox." https://github.com/cgre-aachen/open_AR_Sandbox. Accessed: 2021-03-25.

[2] T. Khan, K. Johnston, and J. Ophoff, "The impact of an augmented reality application on learning motivation of students," *Advances in Human-Computer Interaction*, vol. 2019, 2019.

[3] "Fysisk sandkasse." https://www.researchgate.net/profile/Karen-Vaughan-12/publication/318073192/figure/fig1/AS:669010110709770@1536515848814/Diagram-of-an-augmented-reality-AR-sandbox-A-and-the-AR-sandbox-in-the-Introduct ppm. Accessed: 2021-05-07.

[4] "Numpy array." https://cs231n.github.io/python-numpy-tutorial/#arrays. Accessed: 2021-05-21.

[5] "Jupyter notebook." https://jupyter.org/. Accessed: 2021-03-25.

[6] "Panel." https://panel.holoviz.org/. Accessed: 2021-05-09.

[7] "Bokeh." https://bokeh.org/. Accessed: 2021-05-09.

[8] "Aruco marks." https://imaginghub.com/uploads/ckeditor/pictures/2018/22088185-a833-40a3-a9c4-dfe66fae43c4.png. Accessed: 2021-05-09.

[9] "Aruco detection." https://www.pyimagesearch.com/2020/12/14/generating-aruco-markers-with-opencv-and-python/. Accessed: 2021-05-18.

[10] "Monte carlo method." https://en.wikipedia.org/wiki/Monte_Carlo_method. Accessed: 2021-05-06.

[11] "Dem." https://library.carleton.ca/help/dem-formats. Accessed: 2021-05-23.

[12] "Pygimli." https://www.pygimli.org/about.html. Accessed: 2021-05-24.

[13] "Elektrisk potensial." https://no.wikipedia.org/wiki/Elektrisk_potensial. Accessed: 2021-05-24.

[14] "What is python? executive summary." https://www.python.org/doc/essays/blurb/. Accessed: 2021-03-24.

[15] "Anaconda." https://www.anaconda.com/products/individual. Accessed: 2021-05-18.

[16] "Anaconda pakker." https://docs.anaconda.com/anaconda/packages/py3.8_win-64/. Accessed: 2021-05-21.

[17] "Gempy." https://www.gempy.org/. Accessed: 2021-03-25.

[18] "Pyqt utviklere, riverbank computing." https://riverbankcomputing.com/. Accessed: 2021-05-20.

[19] "Pyqt." https://riverbankcomputing.com/software/pyqt. Accessed: 2021-03-24.

[20] "Numpy." https://numpy.org/. Accessed: 2021-05-07.

[21] "Scipy." https://www.scipy.org/. Accessed: 2021-05-16.

[22] "k-d tree." https://en.wikipedia.org/wiki/K-d_tree. Accessed: 2021-05-16.

[23] "Pyvista." https://www.pyvista.org/. Accessed: 2021-04-07.

[24] "Matplotlib." https://matplotlib.org/. Accessed: 2021-04-07.

[25] "Css." https://developer.mozilla.org/en-US/docs/Web/CSS. Accessed: 2021-05-06.

[26] "Voila." https://blog.jupyter.org/and-voil%C3%A0-f6a2c08a4a93. Accessed: 2021-05-14.

[27] "Kinect for xbox one (2013)." https://en.wikipedia.org/wiki/Kinect#Kinect_for_Xbox_One_(2013). Accessed: 2021-03-25.

[28] "Kinect v2 sensor." https://www.mdpi.com/sensors/sensors-20-01119/article_deploy/html/images/sensors-20-01119-g004.png. Accessed: 2021-05-20.

[29] "Github." https://en.wikipedia.org/wiki/GitHub. Accessed: 2021-04-07.

[30] "Rake (geology)." https://github.com/erikx50/SEB_Addon. Accessed: 2021-05-28.

[31] "Metadata." https://techterms.com/definition/metadata. Accessed: 2021-05-28.

[32] "Os." https://docs.python.org/3/library/os.html. Accessed: 2021-05-20.

[33] "Signal." https://docs.python.org/3/library/signal.html. Accessed: 2021-05-20.

[34] "Qmainwindow." https://doc.qt.io/qt-5/qmainwindow.html#details. Accessed: 2021-05-13.

[35] "Qwidget." https://doc.qt.io/qt-5/qwidget.html#details. Accessed: 2021-05-13.

[36] "Qdialog." https://doc.qt.io/qt-5/qdialog.html#details. Accessed: 2021-05-13.

[37] "Qtoolbutton." https://doc.qt.io/qt-5/qtoolbutton.html#details. Accessed: 2021-05-13.

[38] "Qwebengineview." https://doc.qt.io/qt-5/qwebengineview.html#details. Accessed: 2021-05-13.

[39] "Interpolation approach gempy." https://docs.gempy.org/index.html?highlight=orientation#interpolation-approach. Accessed: 2021-05-18.

[40] "Forkastning." https://no.wikipedia.org/wiki/Forkastning. Accessed: 2021-05-18.

[41] "Diskordans (geologi)." https://snl.no/diskordans_geologi. Accessed: 2021-05-18.

[42] "Theano (software)." https://en.wikipedia.org/wiki/Theano_(software). Accessed: 2021-05-18.

[43] "Visible geology." https://app.visiblegeology.com/. Accessed: 2021-05-18.

[44] "Modulo." https://no.wikipedia.org/wiki/Modulo. Accessed: 2021-05-24.

[45] "Strike og dip image." https://www.rockmass.net/files/strike&dip.png. Accessed: 2021-05-20.

[46] "Strøk og fall." https://no.wikipedia.org/wiki/Str%C3%B8k_og_fall. Accessed: 2021-05-20.

[47] "Fault types image." https://www.sms-tsunami-warning.com/theme/tsunami/img/fault-types.jpg. Accessed: 2021-05-28.

[48] "Rake (geology)." https://en.wikipedia.org/wiki/Rake_(geology). Accessed: 2021-05-28.

[49] "Line plane collision." https://gist.github.com/TimSC/8c25ca941d614bf48ebba6b473747d72. Accessed: 2021-04-12.

[50] "Pandas." https://pandas.pydata.org/. Accessed: 2021-05-20.

[51] "Pickle." https://docs.python.org/3/library/pickle.html. Accessed: 2021-05-20.

[52] "Bat." https://www.reviversoft.com/no/file-extensions/bat. Accessed: 2021-05-20.

[53] "Folding." https://no.wikipedia.org/wiki/Folding. Accessed: 2021-05-23.

Figurliste

Figurer

Kodeliste

Kode

Vedlegg

A User Manual
B Demovideo av programvare
C Kildekode