Aarhus University School of Engineering

Sundhedsteknologi og Informations- og kommunikationsteknologi
Bachelorprojekt

Dokumentation

Automatisk Ultralydsscanner

Charlotte Søgaard Kristensen (201371015)
Mathias Siig Nørregaard (201270810)
Marie Kirkegaard (201370526)

Vejleder
Associate Professor
Michael Alrøe
Aarhus University School of Engineering

16. december

Indholdsfortegnelse

Versionshistorik 1

Version	Dato	Ansvarlig	Beskrivelse
1.0	2016-11-01	CSK	Første version af udviklingsdokuement med systembeskrivelse, bdd og ibd
1.1	2016-11-03	CSK	Sekvensdiagrammer tilføjet
1.2	2016-11-04	CSK, MSK	Forklaringer til sekvens- og klassediagrammer
1.3	2016-12-06	MK, MSN, CSK	Indsat tekst og figurer til sekvens-, pakke-, tilstands- og klassediagrammer, samt IBD og BDD

Tabel 1.1: Versionshistorik

Indledning 2

Dette dokument indeholder arkitektur og design for systemets hardware og software. Formålet med dokumentet er at give et overblik over, hvordan systemet Automatisk Ultralydsscanning er designet og udviklet. Forklaring på forkortelser findes i bilag 22 Sætningsliste.

Systemarkitektur 3

Der er udarbejdet forskellige diagrammer på baggrund af de specificerede systemkrav. Diagrammerne har til formål at dele systemet op i realiserbare dele for at vise arkitekturen for systemet.

Arkitekturen beskriver den grundlæggende organisering af Automatisk Ultralydsscanner og opbygningen af dens tilhørende PC Applikation. Systemet er opbygget generisk, så man vil kunne udskifte komponenter som f.eks. Robotarm eller 3D kamera med en anden type eller model. Udskiftning af komponenter vil dog resultere i en anderledes implementering. Der er i diagrammerne designet ud fra, at 3D kamera er af typen Microsoft Kinect 2.0 og Robotarm er en UR10 robot.

3.1 Systemoversigt

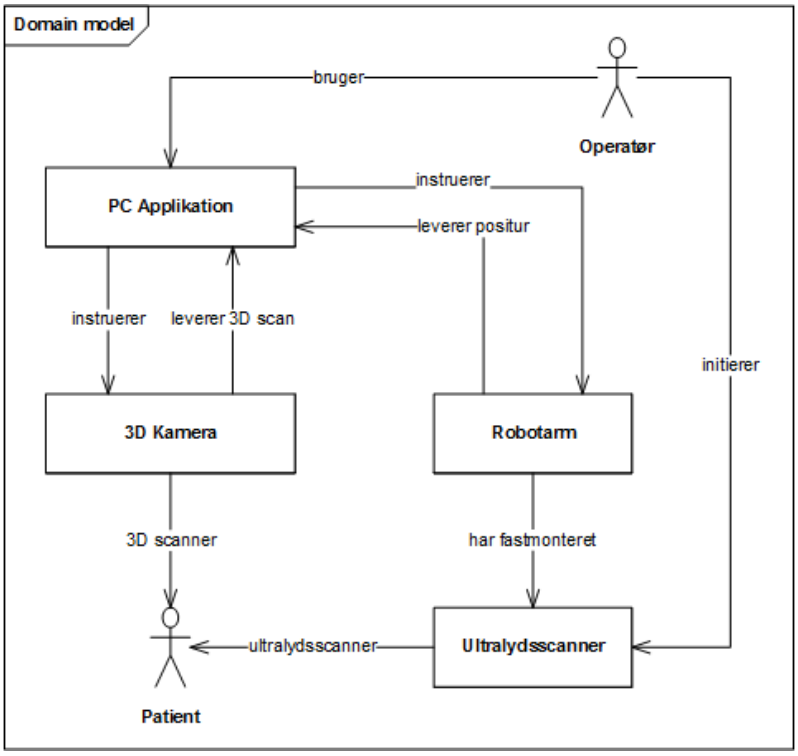
Systemet Automatisk Ultralydsscanner består af en PC applikation, en Ultralydsscanner, en Robotarm af typen Universal Robots UR10, et 3D kamera af typen Microsoft Kinect 2.0, samt et Access Point, af typen D-Link DAP-1160, til forbindelse mellem Robotarm og PC applikation. Der er fem aktører, Robotarm, Ultralydsscanner, 3D kamera, Operatør og Patient, som interagerer med PC Applikation.

Robotarm har en touch skærm, hvorpå Operatør manuelt kan flytte Robotarm, se Robotarms koordinator samt programmere Robotarm. Robotarm er forbundet via et ethernet kabel af typen RJ45 til et Access Point. PC Applikation er forbundet til Access Point med et kabel af samme type. 3D kamera er forbundet til PC Applikation via 3D kameras USB 3.0 kabel. Ultralydsscanner er en seperat enhed, som blot er fastgjort mekanisk på Robotarms yderste led, men indgår i det fulde system Automatisk Ultralydsscanner. Ultralydsscanner skal manuelt tændes og slukkes af Operatør.

For Automatisk Ultralydsscanner er der udarbejdet forskellige diagrammer, som har til formål at dele systemet op i blokke og vise integration mellem blokkene, samt forbindelser mellem aktører. Diagrammerne vil blive gennemgået nedenfor.

3.1.1 Domænemodel

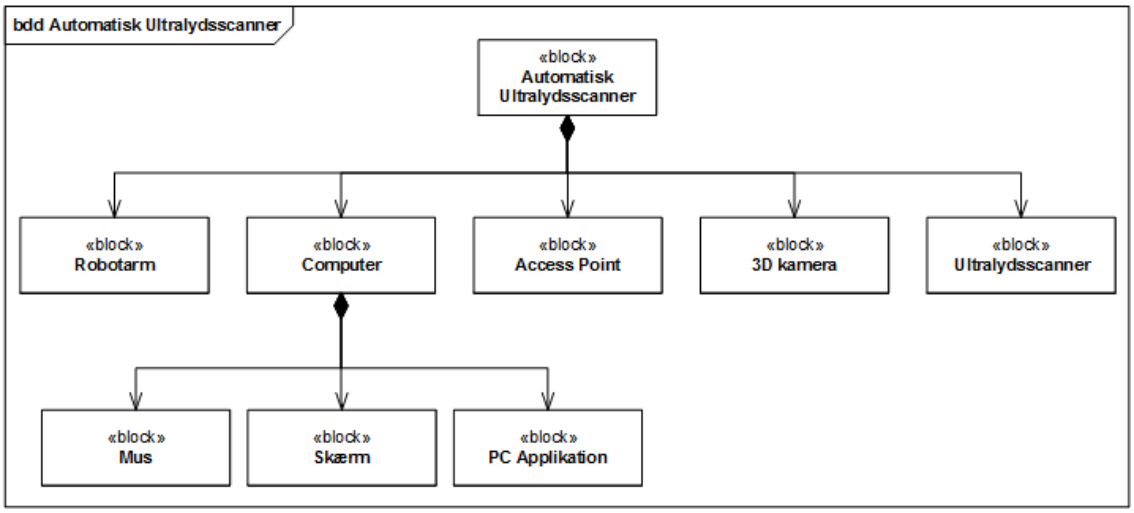
Domænemodellen på Figur 3.1 viser forbindelserne mellem de forskellige aktører i systemet.



Figur 3.1: Domænemodel for Automatisk Ultralydsscanner

3.1.2 Block Definition Diagram

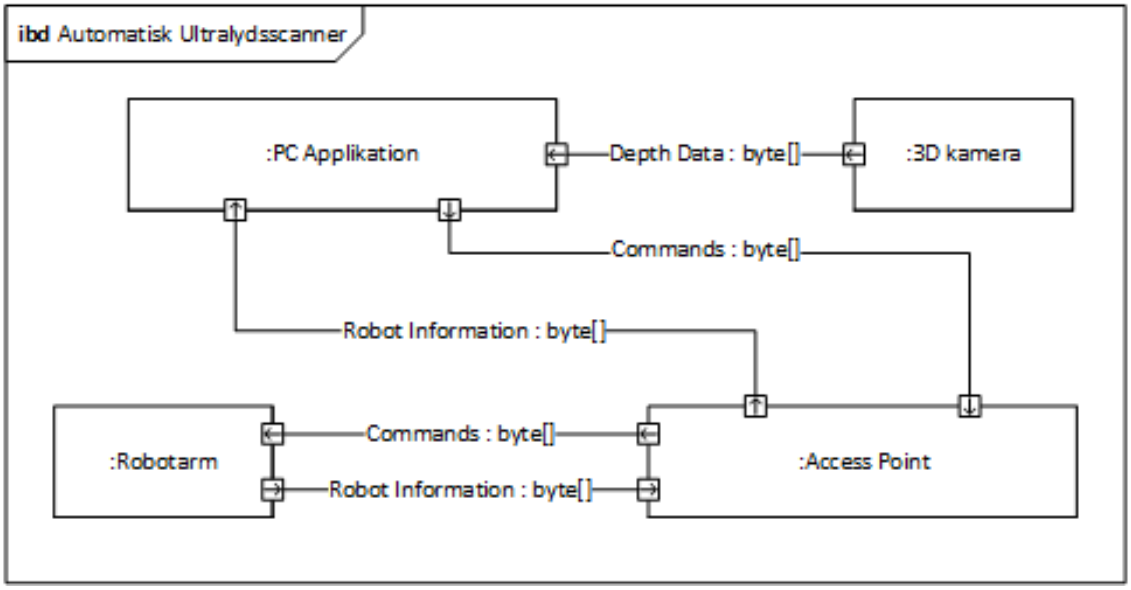
Block Definition Diagram Figur 3.2 giver et overblik over Automatisk Ultralydsscanners komponenter, som et samlet system. Her ses, at det er nødvendigt at have en computer med mus og skærm for at anvende PC Applikation.



Figur 3.2: BDD for Automatisk Ultralydsscanner

3.1.3 Internal Block Diagram

Internal Block Diagram Figur 3.3 viser flow af information mellem de forskellige blokke i Automatisk Ultralydsscanner. Bemærk at Ultralydsscanner ikke er med, da den ikke interagerer med resten af Automatisk Ultralydsscanner, men blot er forbundet mekanisk til Robotarm. Når 3D Scan menuen åbnes i PC Applikation, vil der være et konstant flow af dybdedata fra 3D kamera til PC Applikation. Når PC Applikation startes, vil der være et flow af data frem og tilbage mellem Robotarm og PC Applikation, hvor Access Point agerer som mellemled. Forbindelsen mellem PC Applikation og Access Point, samt forbindelsen mellem Access Point og Robotarm er oprettet med ethernet-kabler af typen RJ45. Forbindelsen mellem PC Applikation og 3D kamera er oprettet med 3D kameras USB 3.0 kabel.



Figur 3.3: IBD for Automatisk Ultralydsscanner

3.2 Systemets grænseflader

Systemet består af to grænseflader: Mellem PC Applikation og Kinect, og mellem PC Aplikation og UR10. Kommunikationen mellem PC Applikation og UR10 sker over modbus-protokollen, hvor kommunikationen mellem PC Applikation og Kinect er gennem Kinect's API via USB.

3.2.1 UR10

Overførsel af data til UR10 sker gennem Transmission Control Protocol/IP-protokollen (TCP/IP). Til afsendelse af positur-værdier fra PC Applikation anvendes modbus-protokollen. Modbus-protokollen sørger for at skrive binære værdier på registre på UR10-controlleren. UR10 kører på URScripts, hvis den skal styres automatisk. UR10 har et script der i en uendelig løkke læser værdierne på registrene og instruerer UR10s Tool Center Point (TCP) til at flytte sig til en positur med en given acceleration og hastighed.

TCP/IP

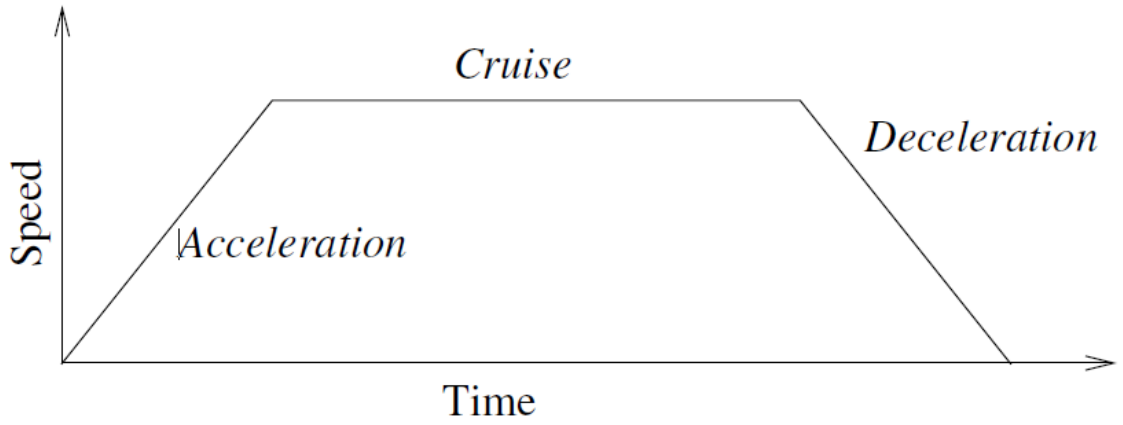
PC Applikation skriver til UR10 over en TCP/IP forbindelse på en IP. Der anvendes to porte; port 502 for kommunikation over modbus-protokollen, og port 30002 for indlæsning af nuværende værdier. Kommunikationen over port 502 er både read og write, hvor port 30002 kun er read. Bemærk, at forskellen ligger i at de værdier der bliver overført på port 502 kun er til styring af UR10 på script-niveau, som fx den ønskede positur, hastighed og acceleration. Modsat, på port 30002, indhentes de nuværende tilstandsværdier som UR10 har, som fx posituren den reélt har, som ikke nødvendigvis er den samme som den sidste ønskede positur. For at give et eksempel på hvordan dette foregår sekvensmæssigt: PC Applikation sender en positur over port 502. Værdierne i denne positur indskrives på UR10s registre. URScriptet aflæser disse værdier og instruerer UR10 i at flytte sit TCP til denne positur. Efter noget tid vil den have nået denne positur. Der vil løbende kunne aflæses om UR10 har nået posituren på port 30002.

Port 502 Der er mulighed for at sende informationer om hvor Robotarm skal bevæge sig hen over denne port. For detaljeret specifikation om port 502 og port 30002 henvises der til TRU's dokumentation (27) styk 2.3.2 på side 6.

Port 30002 Der er mulighed for at hive informationer om stort set alle Robotarms nuværende tilstande som fx software version, ledrotationshastighed, tryk feedback m.m. I implementeringen anvendes der kun TCPs rotation og position.

URScript

Scriptet der kører på UR10 aflæser i alt 3 værdier: Acceleration, hastighed og positur. Positur består af værdierne X, Y og Z for position og RX, RY og RZ for rotation. Positur-værdierne aflæses på registrene 135-140 og indsættes som properties i et 'pose' objekt. Accelerationsværdien, hastighedsværdien samt pose-objektet gives som parameter til funktionen *movel*. UR10s TCP flyttes lineært i funktionen *move1*. Tiden på acceleration og deceleration styres omvendt proportionalt af accelerationsværdien, mens den maksimale hastighed styres af hastighedsværdien. Se figur 3.4 for et billede af dette. Se manuelen for UR10 (bilag 28) s. II-48 til II-50 for mere information om *movel*-kommandoen.



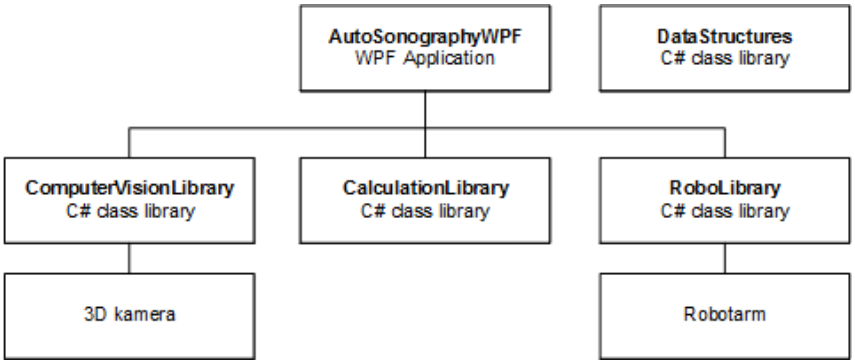
Figur 3.4: Hastighed over tid som vist i UR10s brugermanuel

3.2.2 Kinect

Computeren er forbundet til 3D kamera via USB af typen Transcend PDU3 USB-adapter PCIe 2.0 USB 3.0 eller anden USB3.0 type, der er kompatible med Microsoft Kinect 2.0. 3D kamera har også et supplerende strømforsyningskabel. For kommunikation med 3D kamera er der brugt Microsoft's Kinect API [**?**] . Der er taget udgangspunkt i Fusion-delen [**?**] af API'et. Fordelen med dette API er at det kan gøre en masse arbejde - ulempen er at det ikke er generisk. Altså ville man ikke kunne bruge en anden sensor en Microsoft Kinect for Windows 2.0. API'et har en C#-wrapper, og Microsoft har leveret kildekode til et projekt der gjorde nogenlunde det vi var interesserede i (se reference [**?**] for at få den nyeste version af dette. Efter oprettelsen af forbindelse til Kinect-sensoren er det muligt at 'lytte' på den og få de depth frames den kontinuerligt leverer. Se beskrivelsen af sekvensdiagrammet 3D scan (figur 4.6) i Systemdesign for yderligere forklaring.

3.3 Softwarearkitektur

PC Applikation er opdelt af forskellige moduler for at øge samhørigheden og nedsætte koblingen, hvilket er med til at sikre overskuelighed og gøre PC Applikation lettere at vedligeholde og genbruge. Modulerne er inddelt efter ansvarsområder angående præsentation til bruger, kommunikation til Robotarm, indhentning af 3D scan fra 3D kamera samt udregning af Robotarms sti til ultralydsscanning. Disse 4 moduler har fælles datastrukturer. For at undgå cykliske forbindelser, er disse datastrukturer tilføjet til deres eget modul. Figur 3.5 viser referencen mellem modulerne.



Figur 3.5: PC Applikations opdeling af moduler

Lag	**Beskrivelse af ansvar**
AutoSonographyWPF	Håndterer Operatørs interaktion med PC Applikation, hvor Operatør kan vælge 3D scan- eller ultralydsscan. Dette projekt virker som en grænseflade til resten af PC Applikation.
ComputerVisionLibrary	Indhenter og afgrænser 3D scanning fra 3D kamera.
CalculationLibrary	Sørger for at konvertere en 3D scanning til en sti af positurer som Robotarm kan gennemløbe.
RoboLibrary	Muliggør at kommunikere med og instruere Robotarm.
DataStructures	Indeholder forskellige data transfer objekter (DTO), der bruges gennem PC Applikation til at sende objekter mellem de forsellige moduler, samt udvidelsesmetoder til eksisterende .NET eller KinectAPI datastrukturer.

Tabel 3.1: Modulopdeling og ansvar

3.3.1 Pakkediagram

Pakkediagrammet figur 3.6 giver en oversigt over afhængighederne i PC Applikation. For at undgå cykliske forbindelser blev adskillige datastrukturer flyttet fra CalculationLibrary, ComputerVisionLibrary og RoboLibrary til DataStructures-biblioteket. DataStructures indeholder altså kun nogle datastrukturer og nogle extension-metoder til disse. For forklaringen af indholdet af de øvrige pakker, se klassediagrammerne for hvert bibliotek.

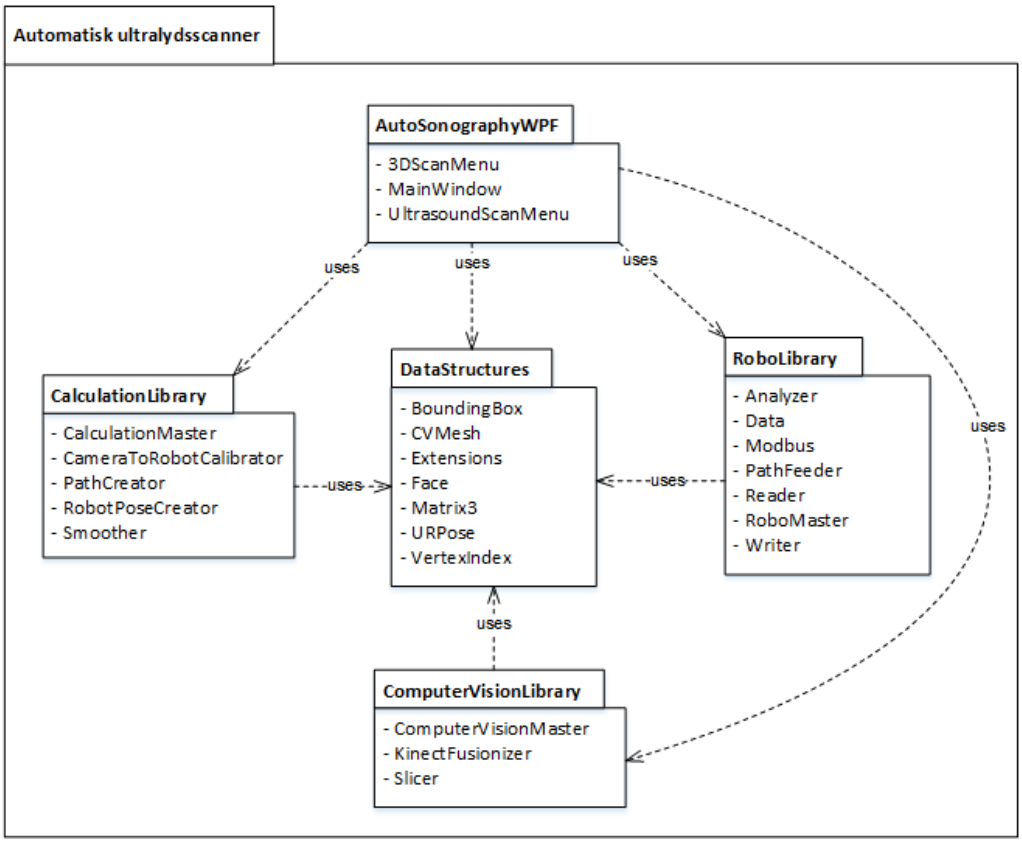
AutoSocograophyWPF inderholder alt, der hører til PC Applikations grafiske brugergrænseflade (GUI).

ComputerVisionLibrary indeholder klasser, der interagerer med 3D kamera og udvinder en 3D scanning i et begrænset område.

RoboLibray indeholder klasser som der muliggør kommunikation med Robotarm.

DataStructures indeholder datastrukturer der bruges på tværs af PC Applikation og nogle extension-metoder til disse.

CalculationLibrary indeholder klasser der finder stien af positurer i en 3D scanning der er nødvendige for at foretage en ultralydsscanning med Robotarm.



Figur 3.6: Pakkediagram for Automatisk Ultralydsscanner

Systemdesign 4

Der er udarbejdet forskellige diagrammer på baggrund af de specificerede systemkrav. Diagrammerne har til formål at dele systemet op i realiserbare dele for at vise designet af systemet.

4.1 Klassediagram

Dette afsnit beskriver klasserne fra pakkediagrammet. Klassediagrammerne viser strukturen i systemet og deres relationer. Hver klasse indeholder de vigtigste metoder og attributer i klassen, der udgør funktionaliteten i PC Applikation.

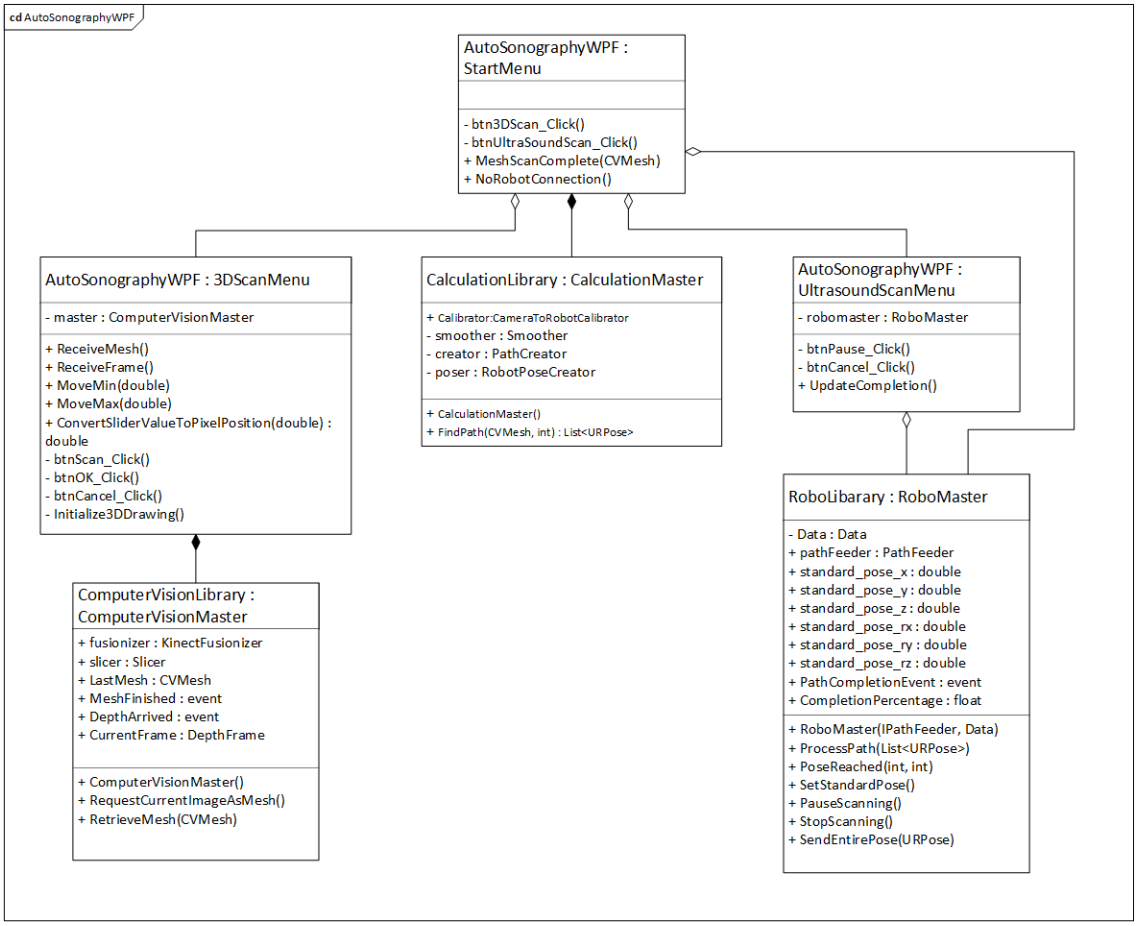
4.1.1 GUI

Denne klasse indeholder brugergrænsefladen af PC Applikation.

- **MainWindow**
  Giver anledning til at foretage et 3D scan. Såfremt en 3D scanning er gennemført, giver det også anledning til at starte en ultralydsscanning. Når menuen startes, oprettes en instans af RoboMaster, for at sætte Robotarm i standard positur. Dette er nødvendigt, hvis Robotarm skulle være i vejen for en 3D scanning. Hvis der ikke er nogen forbindelse til Robotarm vil der komme en prompt med en besked om dette.

- **3DScanMenu**
  I denne menu er der mulighed for at se det nuværende dybdebillede, afgrænse området der skal 3D scannes og foretage en 3D scanning.

- **UltrasoundScanMenu**
  I denne menu kan den procentvise færdiggørelse af ultralydsscanningen følges. Der er også mulighed for at pause samt afbryde ultralydsscanningsprocessen.

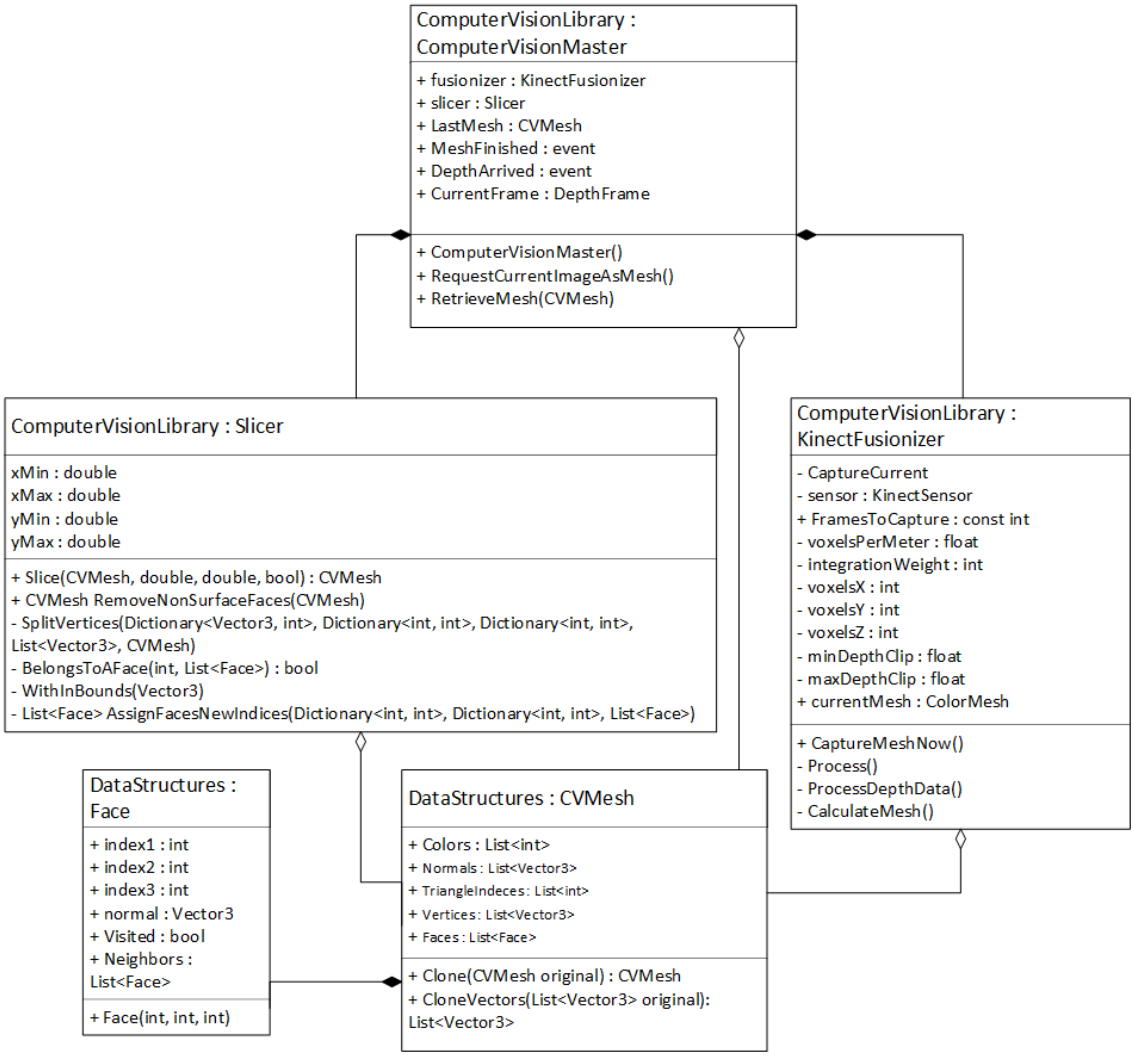


Figur 4.1: Klassediagram for GUI

4.1.2 ComputerVisionLibrary

Formålet med dette bibliotek er at få en afgrænset 3D scanning fra et 3D kamera.

- **KinectFusionizeren**
  Har til ansvar at åbne Kinect-sensoren, tage det nuværende dybdebillede fra sensoren og konvertere det til en mesh.
- **ComputerVisionMaster**
  Denne klasse virker som den logiske grænseflade til KinectFusionizeren, hvor instansen af den nuværende mesh lagres her. Andre klasser kan subscribe til ComputerVisionMasteren for at høre, hvornår der er en ny mesh tilgængelig.
- **Slicer**
  Denne klasse sørger for at fjerne de punkter i en mesh, der er uinteressante:
  1. Faces der peger ned af, dvs fejlpunkter. Da 3D kamera er monteret i loftet, vil den ikke kunne se undersiden af det den scanner.
  2. Duplikerede punkter. KinectFusionizer outputter punkter der er ens. Disse fjernes af optimeringsårsager.
  3. Punkter og faces der er uden for det område der ønskes scannet. Dette inkluderer nærtliggende objekter som fx en væg, eller områder på patienten der ikke ønskes scannet.



Figur 4.2: Klassediagram for ComputerVisionLibrary

4.1.3 CalculationLibrary

Dette bibliotek agerer som bindeledet mellem ComputerVisionLibrary og RoboLibrary.

- **CameraToRobotCalibrator**
  Sørger for at konvertere 3D scanningen givet fra ComputerVisionLibrary fra 3D Kameras rum til Robotarms rum. Dette sker ved en kæde af matrix-transformationer i en speciel rækkefølge. Normaltvis har man en translation, rotation og skalering, men da Robotarms og 3D Kameras koordinatsystemer begge er angivet i millimeter, er skaleringen unødvendig. I tilfældet for dette projekt sker der først en rotering og derefter en translation, for at bestemme transformationsmatricen. Hvert punkt i en mesh konverteres så til det nye space. Se reference [**?**] for inspirationen til denne klasse.

- **Smoother**
  Denne klasse har til ansvar at udjævne en mesh. Med udjævning forstås, at ensrette retningsvektorerne i en mesh's faces. Dette vil bivirke at ved en scanning af et relativt flat objekt og N antal smoothing-runs, vil meshens faces være stort set ens. Dette er nødvendigt da 3D kameras output kan indeholde anomalier og dermed vil nogle normaler på 3D scanningens overflade være ekstreme/deforme. Med ekstreme normaler menes der normaler der ikke ligner dens nærmeste nabo-normaler på meshen. Disse anomalier kan bivirke at Robotarm roteres tilsyneladende forkert imod meshen. Udjævningen sker gennem laplacian smoothing hvor alle retningsvektorer i overfladen ensrettes. Se [1] for forklaring af anvendt algoritme.

- **PathCreator**
  Klassen afgør listen af punkter i en mesh som der skal findes positurer til Robotarm ud fra. For at afgøre stien genereres der en 'bølge' - i implementeringen en squarewave - af punkter der draves over meshen. De vertices i meshen der tilnærmer sig punkterne i bølgen bedst vil blive udvalgt til stien.

- **RobotPoseCreator**
  I denne klasse vil konverteringen af en mesh-sti til en liste af positurer ske. For hvert punkt i mesh-stien, vil en vertex' normal findes. Ved hjælp af normalen, sti-punktets koordinater samt længden på Robotarms probe kan den forskudte Robotarm position findes. Inverteres denne normal, kan det ses som en retningsvektor for en Robotarm. Retningsvektoren konverteres først til en roll, pitch og yaw - altså roteringer omkring de tre retningsakser; X, Y og Z. Da man ikke kan afgøre alle tre værdier ud fra en retningsvektor alene, sættes pitch til 0, da det er muligt at 'pege' et vilkårligt sted med en roll-rotering og en yaw-rotering. Disse værdier konverteres herefter til en rotationsvektor. Positionsvektoren og rotationsvektoren udgør til sammen en positur, som tilføjes til listen af positurer. Matematikken for udregningen af rotationen kan ses på næste side.

RobotPoseCreator rotationsmatematik

Givet en tredimensionel retningsvektor med en længde på 1

$$v_{direction} = (X, Y, Z)$$

Find de tre rotationer om de tre forskellige akser. Med en retningsvektor alene kan én af disse rotationer ikke findes. Da man vil kunne pege i en vilkårlig retning med en roll-rotering og en yaw-rotering, er pitch sat til 0.

$$roll = acos(Z) \qquad pitch = 0 \qquad yaw = \begin{cases} -acos(-Y) & X \leq 0 \\ acos(-Y) & X > 0 \end{cases}$$

Opstil matricerne der skal bruges for at konvertere roll, pitch og yaw til en rotationsvektor.

$$Roll_M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(roll) & -sin(roll) \\ 0 & sin(roll) & cos(roll) \end{bmatrix} \quad Pitch_M = \begin{bmatrix} cos(pitch) & 0 & sin(pitch) \\ 0 & 1 & 0 \\ -sin(pitch) & 0 & cos(pitch) \end{bmatrix}$$

$$Yaw_M = \begin{bmatrix} cos(yaw) & -sin(yaw) & 0 \\ sin(yaw) & cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ordnen af rotationerne er vigtig, derfor vil der først foregå en roll rotering, en pitch rotering og til sidst en yaw rotering. Prikken mellem matricerne her er dot-produktet.

$$\mathbb{R} = Yaw_M \cdot Pitch_M \cdot Roll_M$$

Dernæst kan θ og μ findes, der bruges til beregningen af r_x, r_y samt r_z, der samlet udgør den endelige rotationsvektor $v_{rotation}$.

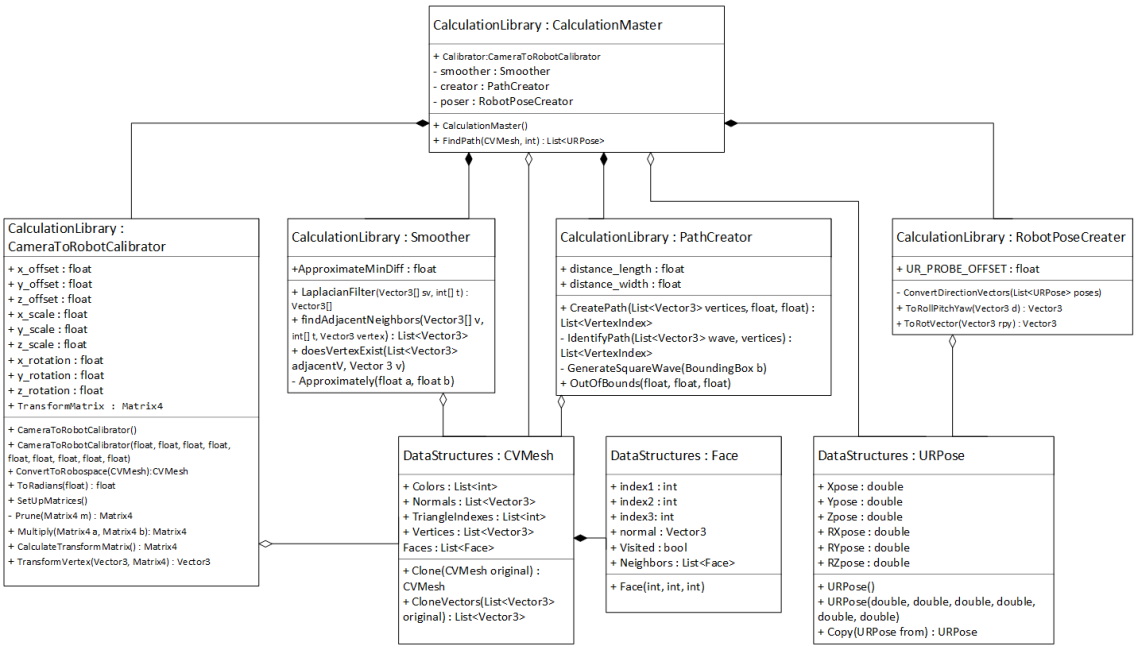
$$\theta = acos\left(\frac{\mathbb{R}_{0,0} + \mathbb{R}_{1,1} + \mathbb{R}_{2,2} - 1}{2}\right) \qquad \mu = \frac{1}{2 \times sin(\theta)}$$

$$r_x = \mu \times (\mathbb{R}_{2,1} - \mathbb{R}_{1,2}) \times \theta$$

$$r_y = \mu \times (\mathbb{R}_{0,2} - \mathbb{R}_{2,0}) \times \theta$$

$$r_z = \mu \times (\mathbb{R}_{1,0} - \mathbb{R}_{0,1}) \times \theta$$

$$v_{rotation} = (r_x, r_y, r_z)$$

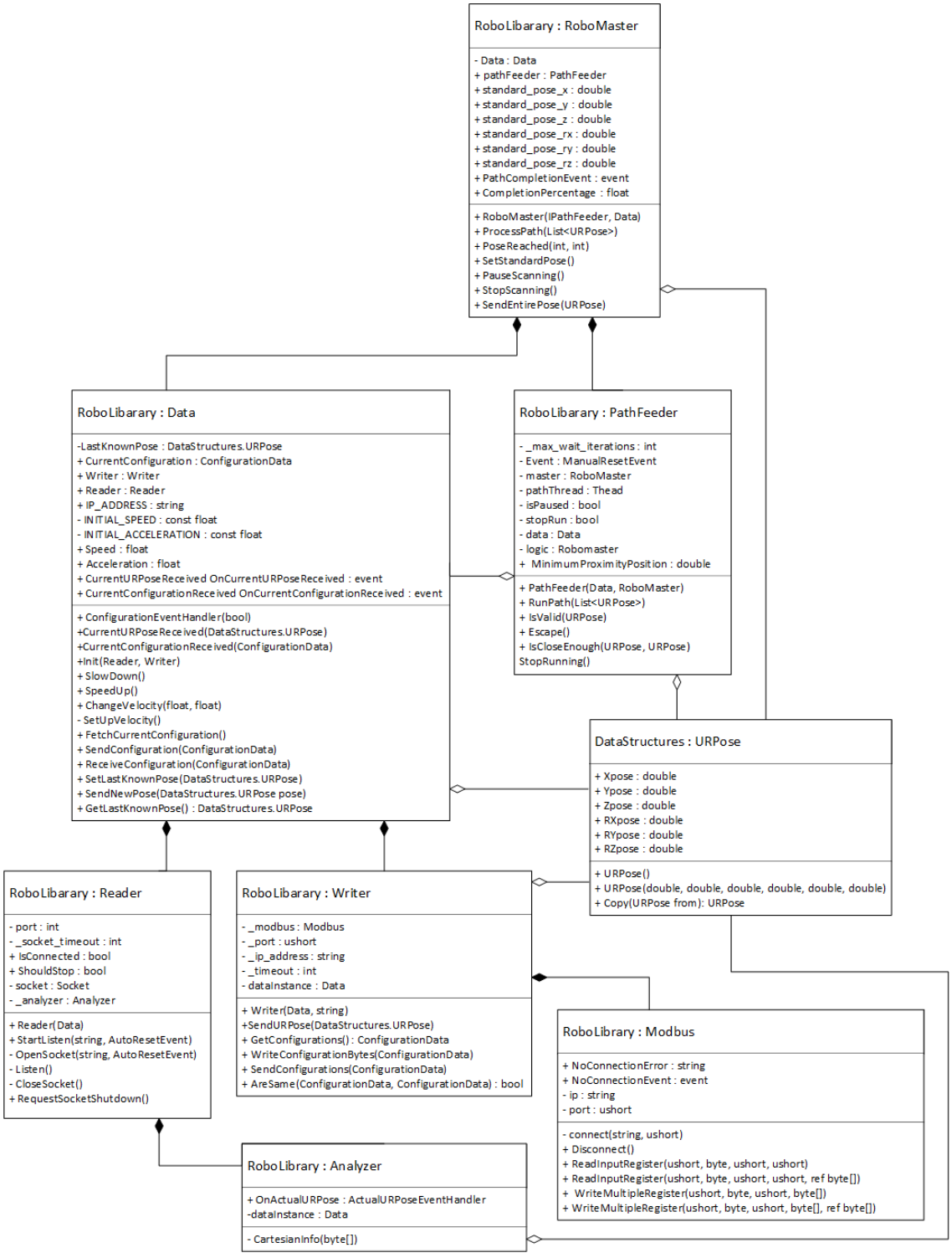


Figur 4.3: Klassediagram for CalculationLibrary

4.1.4 RoboLibrary

Biblioteket giver mulighed for kommunikation med Robotarm.

- **RoboMaster**
  Klassen agerer som bindeled mellem de øvrige klasser i biblioteket og GUI.
- **PathFeeder**
  Står for at gennemløbe hver positur i listen, og kommunikere med Data for at finde ud af hvornår den næste positur skal sendes til Robotarm.
- **Data**
  Klassen virker som en grænseflade mellem den 'logiske' del af biblioteket og dens underliggende reader/writer klasser.
- **Reader**
  Denne klasse står for kontinuerligt at læse data fra Robotarm, for at afgøre dens nuværende positur. Denne klasse er kopieret og ændret fra det tidligere bachelorprojekt, TRU.
- **Analyzer**
  Klassen konverterer det indlæste data til en objekt-orienteret model, altså transformation af bytes til Robotarms nuværende positur. Denne klasse er kopieret og ændret fra det tidligere bachelorprojekt, TRU.
- **Writer**
  Klassen har til ansvar at omskrive værdier til binær data. Den omskriver både positurer samt konfigurationer. Denne klasse er kopieret og ændret fra det tidligere bachelorprojekt, TRU.
- **Modbus**
  Denne klasse skriver binær data ud på Robotarms IP gennem modbus-protokollen. Denne klasse er kopieret og ændret fra det tidligere bachelorprojekt, TRU.



Figur 4.4: Klassediagram for RoboLibrary

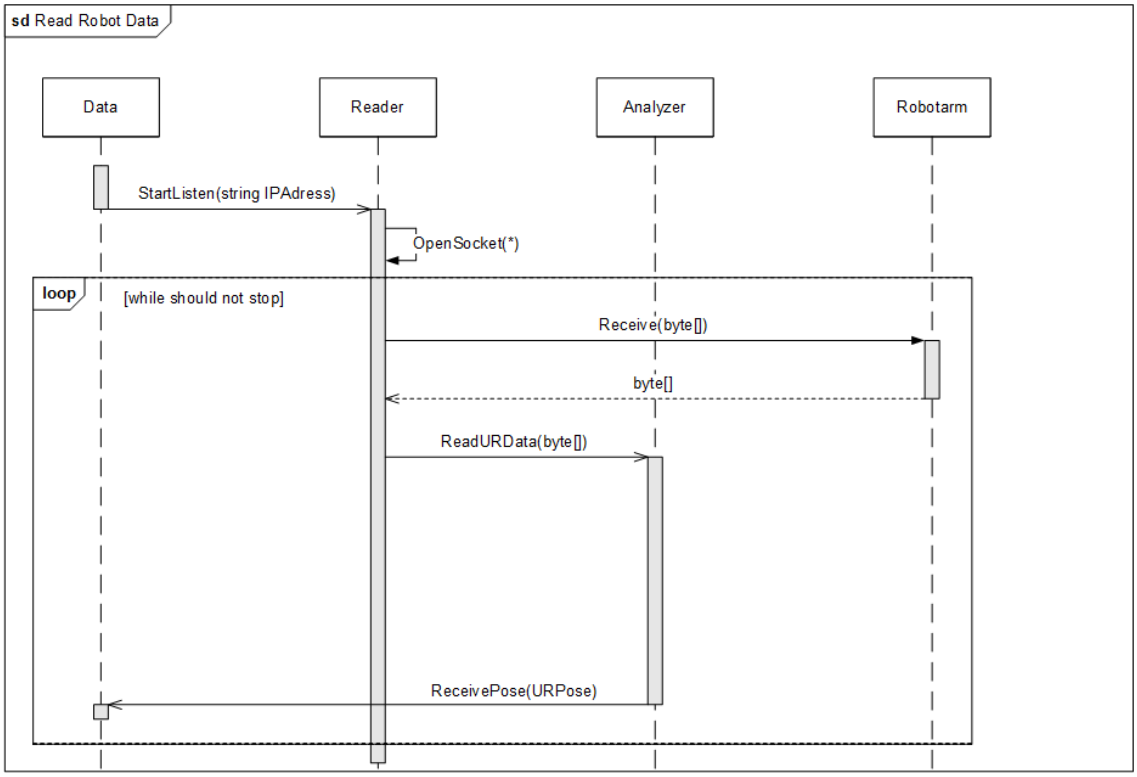
4.2 Sekvensdiagrammer

Der er på baggrund af klassediagrammerne lavet sekvensdiagrammer, som beskriver systemets funktionalitet, og hvor de vigtigste metoder og attributter imellem klasserne er identificeret.

Nedenstående sektioner vil beskrive de vigtigste sekvensdiagrammer i Automatisk Ultralydsscanner og fremvise, hvordan klasserne indbyrdes kommunikerer. Bemærk at for Read Robot Data samt Feed Path er Access Point undladt, da den blot videregiver informationen til Robotarm.

4.2.1 Read Robot Data

Reader initieres med en IP, hvor den skal lytte på. Der åbnes en socket på denne IP, og derefter lytter den kontinuert i en baggrundstråd. Readeren giver de rå data videre til Analyzer som konverterer dem til Robotarms nuværende positur.

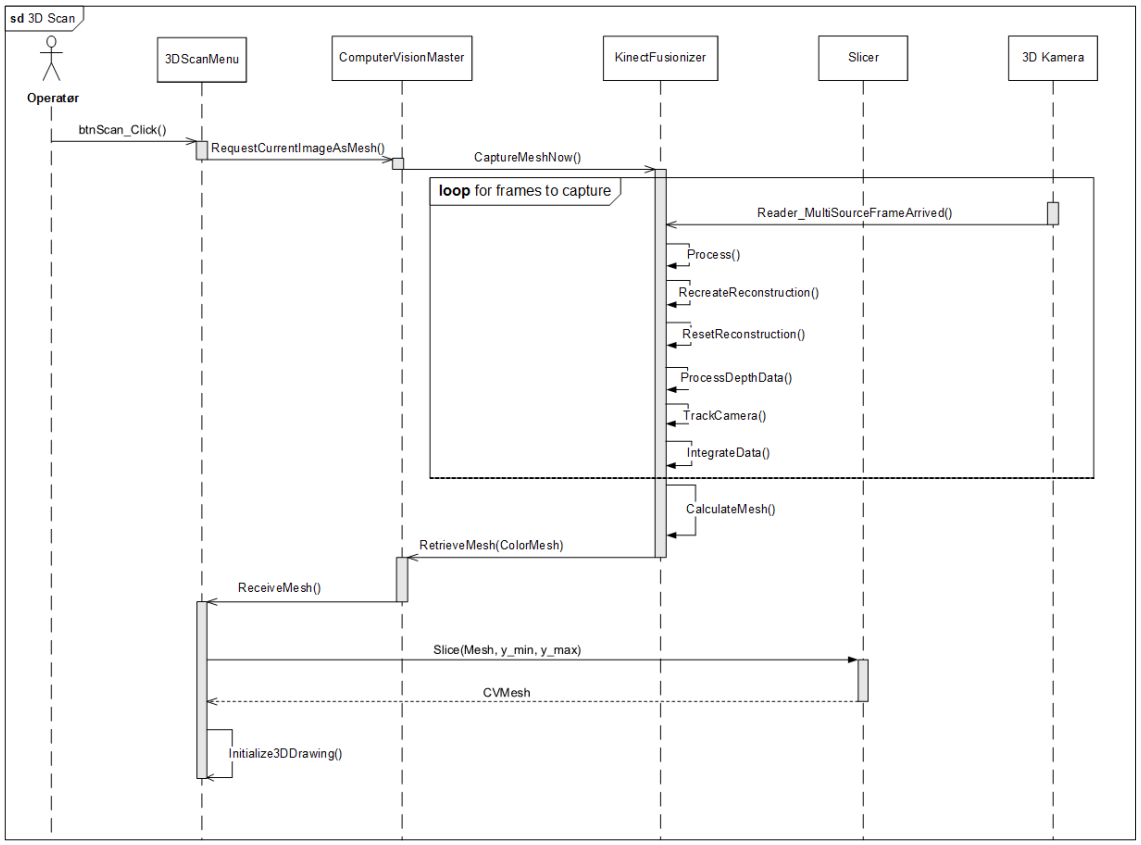


Figur 4.5: Sekvensdiagram for Read Robot Data - indlæsning af Robotarms positur

4.2.2 3D scan

Ved 3D scan vil KinectFusionizeren stå for at få en 3D scanning fra Kinect-sensoren. I denne klasse sørges der for at konvertere et dybdebillede (en depth-frame) fra Kinect's infrarøde kamera til en point cloud. Der afventes det næste dybdebillede, hvis point cloud integreres med det forrige billede. Når der er tilstrækkelige point clouds skal det konverteres til en 3D model. Disse point clouds trianguleres, så der fås en mesh der efterfølgende kan bearbejdes. 3DScanMenu modtager meshen gennem event-kommunikation. Gennem de beskræringsparametre der er valgt i menuen vil meshen dernæst beskæres i klassen Slicer,

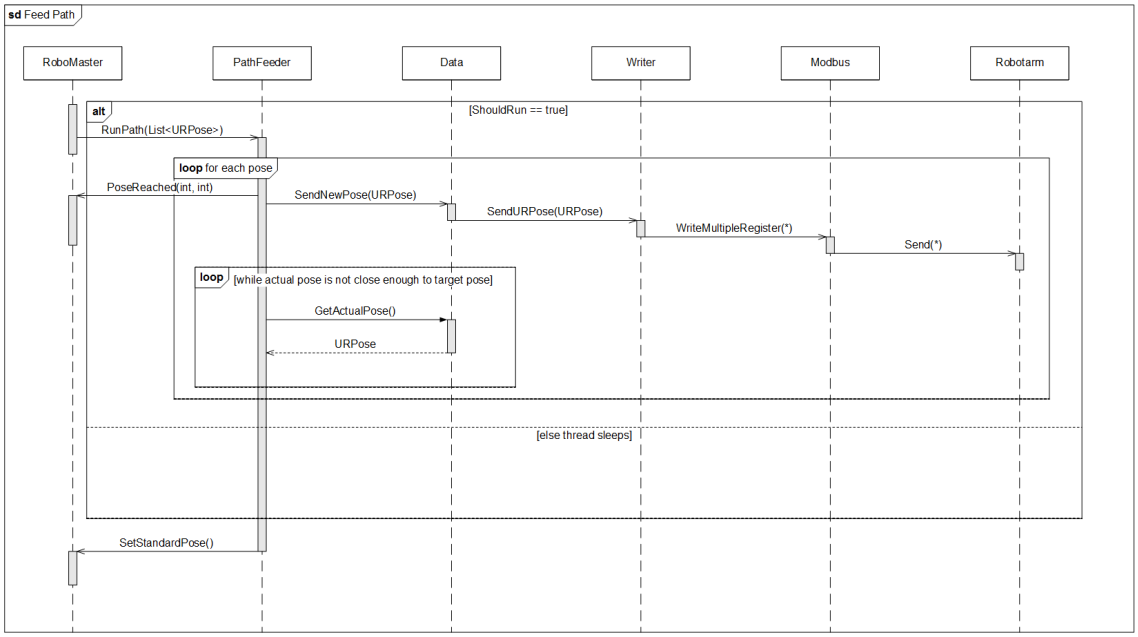
så kun det ønskede område fremkommer. Efter beskæringen vises meshen som en rotérbar 3D model i menuen.



Figur 4.6: Sekvensdiagram for 3D scan

4.2.3 Feed Path

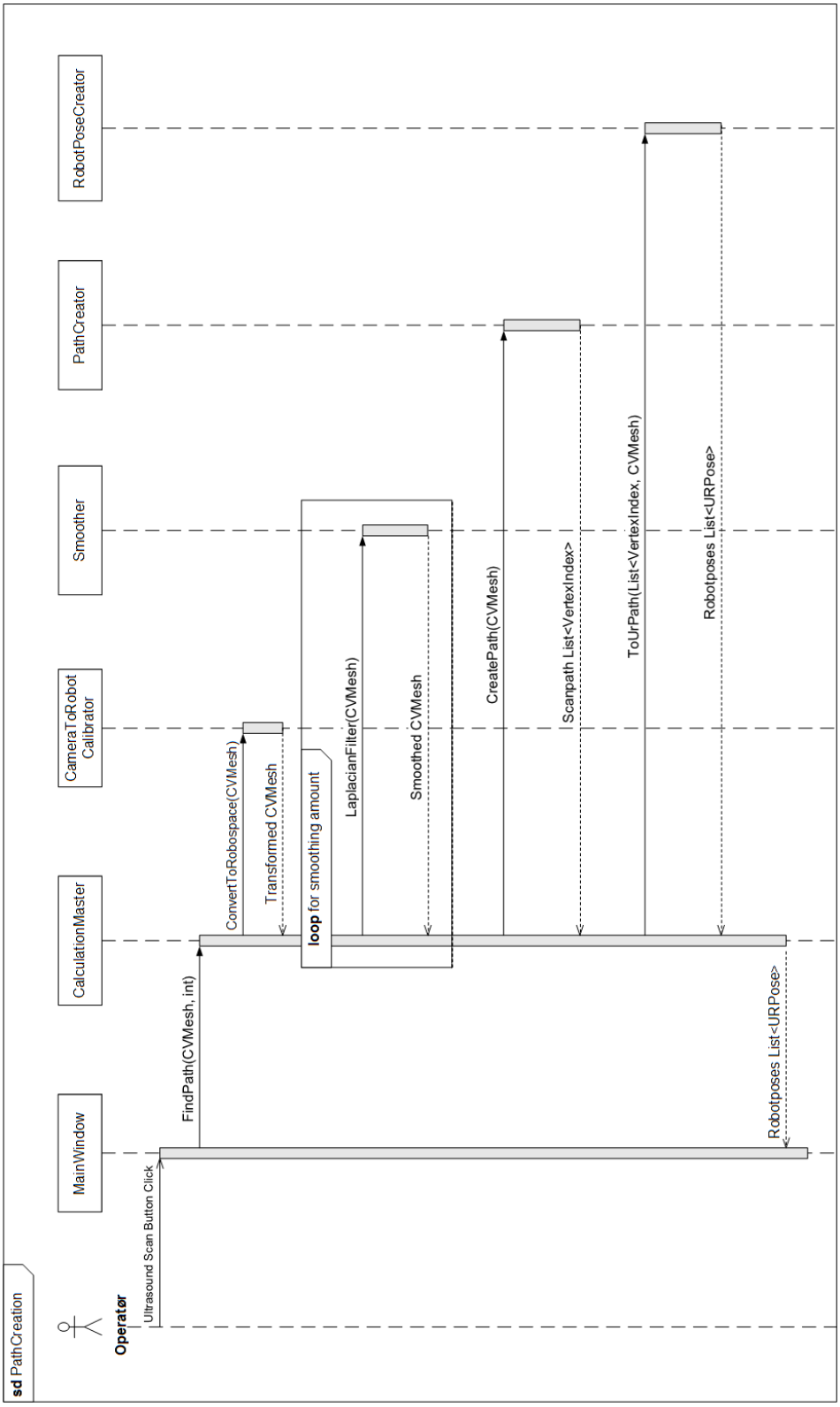
Efter konvertering af mesh output til robot positur sti, vil listen af positurer sendes fra RoboMaster til PathFeeder. Se sekvensdiagrammet 'Path Creation' for en gennemgang af hvordan disse positurer findes. PathFeeder gennemløber hver positur og sender den næste i listen til Data, som videregiver posituren til Writer. RoboMaster informeres løbende om hvor langt PathFeeder er med at gennemløbe punkter. Writer konverterer posituren til binær data, og ModBus skriver dataen ud på Robotarms register. Hernæst ser PathFeeder på om Robotarms nuværende positur har nærmet sig den ønskede positur. Når den er tæt nok på, hoppes der ud af 'while'-løkken, og den næste positur kan sendes. Den Alt der er her skal forstås som at PathFeeder kører i sin egen baggrundstråd der kan pauses. Ved terminering af denne baggrundstråd vil PathFeeder stoppe med at videregive nye positurer til Robotarm.



Figur 4.7: Sekvensdiagram for Feed Path

4.2.4 Pathcreation

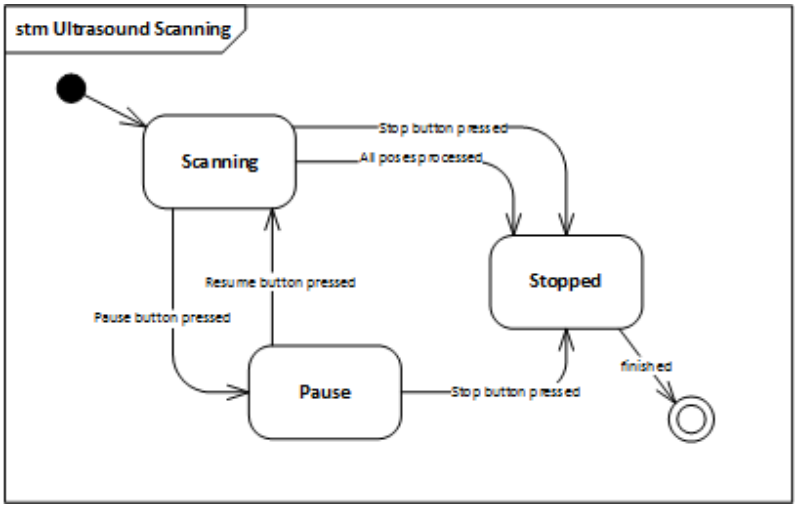
Ved tryk på Ultralydsscan-knappen sendes den scannede mesh videre til CalculationMaster. Se sekvensdiagrammet '3D scan' for at se hvordan den scannede mesh opnås. Først skal meshen konverteres fra 3D Kameras rum til Robotarms rum, dette sker i CameraToRobotCalibrator, hvor meshen roteres og translateres ift. hvor 3D Kamera befinder sig i virkeligheden. Efter transformationen skal meshen udjævnes, så de rå og ekstreme normaler i meshen udglattes. Loopet afgør hvor mange gangen meshen skal gennemkøres filteret. Dernæst sendes den konverterede mesh til PathCreator så der oprettes en liste af de punkter i meshen vi ønsker at Robotarm skal gennemgå. Til sidst skal stien der findes i PathCreator konverteres til Robotarm positurer, da stien i PathCreator kun fortæller position direkte på meshen. Med denne sti får Robotarm afdækket overfladen på Patient, hvis den har en Ultralydsscanner monteret. Stien kan nu videresendes til RoboMaster - se sekvensdiagrammet 'Feed Path' Figur 4.7.



Figur 4.8: Sekvensdiagram for Pathcreation

4.3 Tilstandsdiagram

Dette afsnit beskriver adfærden i systemet ved brug af et tilstandsdiagram. Tilstandsdiagrammet beskriver overgange mellem forskellige tilstande. I UC3: Ultralydsscan brystområde kan Operatør vælge at pause scanningen midlertidig og enten genoptage eller helt stoppe scanning. Figur 4.9 beskriver Robotarms forskellige tilstande under udførelse af ultralydsscanning.



Figur 4.9: Tilstandsdiagram for ultralydsscanning

Udviklingsmiljø 5

Der er i nedenstående Tabel 5.1 opstillet de programmer og versioner der er brugt til udviklingen af Automatisk Ultralydsscanner.

Emne	**Version**
Windows Education	10.0.14393
Microsoft Visual Studio Community	2015
.NET Framework	4.5.2
JetBrains ReSharper Ultimate	2016.2.2
MeshLab	1.3.3
Notepad++	7.2.1
Unity3D	5.4.1
Google SketchUp	16.1
MathCad	14
Microsoft Kinect API	1.8
UR10	3.1.18024
GitHub	2016
SourceTree	1.8.3.0

Tabel 5.1: Udviklingsmiljø

Til operativsystem (OS) er der anvendt Windows, da .NET-frameworket og Microsoft's Kinect API virker natively på dette OS.

I projektet blev Visual Studio og udviklingsframeworket .NET (i sproget C#) anvendt til at udvikle PC Applikation. For at beregne code coverage er der anvendt ReSharper.

Til debugging af 3D scanninger er MeshLab og Notepad++ anvendt.

Google SketchUp er et 3D moduleringsprogam, som i projektet er blevet anvendt til at visualisere de beregninger der skulle til for at rotere Robotarm.

Til 3D visualisering af Automatisk Ultralydsscanner blev Unity anvendt. Dette gjorde det hurtigere at få at overblik over 3D kameras position-offset ift. Robotarm.

Til matematiske beregninger af Robotarms rotation blev Mathcad anvendt.

Til versionsstyring er der brugt GitHub som repository-hosting og SourceTree til git-grænseflade.

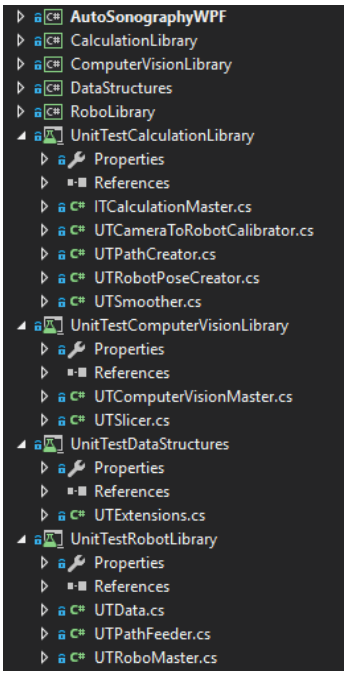
Test 6

I dette afsnit omtales test af PC Applikation. Som udgangspunkt er der lagt fokus på unit-tests. Der er kun lavet én integrationstest, som er en integrationstest af hele CalculationLibrary-projektets indhold. Det vil ikke være muligt at lave en integrationstest, hvor man bruger de lavestliggende moduler i hverken RoboLibrary eller ComputerVisionLibrary, da de hver især er afhængige af hardware. Klasserne der afhænger af hardware er derfor ikke unit-testet og er ekskluderet fra code coverage (CC). Efter unit-testen blev godkendt, er der kort tid efter, blevet udført system-test.

Da der ikke blev fundet nogen fornuftig måde at teste AutoSonographyWPF er den ej heller blevet testet, og er også udelukket fra CC.

6.1 Test-specificering

Hvert bibliotek i solutionen har et tilsvarende test-bibliotek. Hver testbar klasse i et bibliotek har en tilsvarende testklasse. Se figur 6.1 for et overblik over testbibliotekerne. I tilfælde af at en klasse er afhængig af andre klasse, indsættes søm i form af interfaces. Dertil vil afhængighederne mockes ud i testene vha. dummy-klasser.



Figur 6.1: Solution struktur som set i Visual Studio's Solution Explorer

6.1.1 Testklasse-opbygning

Hver testklasse er erklæret med et 'TestClass' tag. Disse testklasser har en 'TestInitialize' metode der køres inden hver 'TestMethod' metode. I test-initialiseringen gennemkøres erklæringer der er fælles for alle test-metoderne. Hver test-metode er navngivet som 'MetodeDerTestes_HvadDerGøres_HvadDerForventes'. I test-metoderne bruges der 'Assert' for at verificere at de faktiske resultater der findes frem til stemmer overens med de forventede resultater. Her er et udpluk af unit-testen for klassen Slicer:

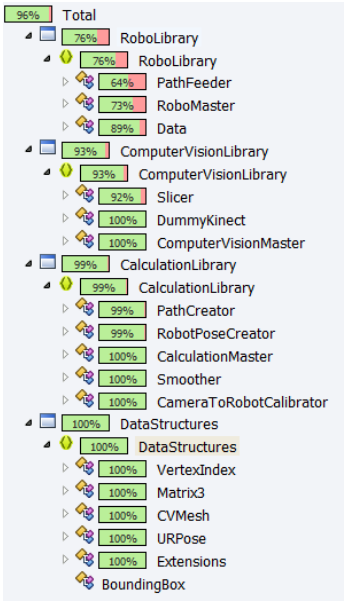
```
...
namespace UnitTestRobotLibrary
{
        [TestClass]
        public class UTSlicer
        {
                private Slicer uut;
                private string testModelLocation;
                [TestInitialize]
                public void Setup()
                {
                        uut = new Slicer();
                        testModelLocation = Directory.GetParent(
                                Directory.GetParent(
                                Directory.GetParent(
                                Environment.CurrentDirectory).
                                ToString()).
                                ToString()) +
                                "\\TestModels";
                }

                [TestMethod]
                public void Slice_InsertFourFaces_2Returned()
                {
                        string location = testModelLocation + @"\fourTriangles.ply";
                        CVMesh mesh = PLYHandler.ReadMesh(location);
                        float lowerLimit = 58.5f;
                        float upperLimit = 1000f;
                        Slicer.xMin = float.MinValue;
                        Slicer.xMax = float.MaxValue;
                        CVMesh sliced = uut.Slice(mesh, lowerLimit, upperLimit, false);
                        Assert.AreEqual(5, sliced.Vertices.Count);
                        Assert.AreEqual(0, sliced.Faces[0].index1);
                        Assert.AreEqual(1, sliced.Faces[0].index2);
                        Assert.AreEqual(2, sliced.Faces[0].index3);
                        Assert.AreEqual(3, sliced.Faces[1].index1);
                        Assert.AreEqual(2, sliced.Faces[1].index2);
                        Assert.AreEqual(4, sliced.Faces[1].index3);
                }
                ...
        }
}
```

I metoden 'Setup' erklæres stien for mappen hvor de modeller som unit-testen skal teste på ligger, og unit under test (uut) oprettes på ny. Der tjekkes med Assert om resultaterne er korrekte. Såfremt at en af 'AreEqual'-metoderne returnerer 'false', vil testen fejle.

6.2 Resultater

Der blev skrevet 42 tests som alle gik igennem pr d. 7/12/2016. Se bilag 24 Testresultater for en oversigt over disse tests og deres resultater.

Disse 42 tests resulterede i en code coverage på 96%. Se figur 6.2 for overblik, og se bilag 4 Code Coverage for en interaktiv navigering af code coveragen.



Figur 6.2: Code coverage udregnet af ReSharper-værktøjet

Code coveragen siger noget om hvor stor en procentdel af den kode, der er valgt at blive inkluderet, som er blevet gennemløbet af unit-tests. Det vil sige at nogle klasser er blevet eksluderet fra denne coverage. Disse elementer fra kravspecifikationen er ikke blevet 'coveret' eller testet:

- **AutosonographyWPF** - Kræver simulering af musetryk m.m.
- **ComputerVisionLibrary:KinectFusionizer** - Kræver tilslutning af hardware
- **RoboLibrary:Analyzer** - Kræver tilslutning af hardware
- **RoboLibrary:ModBus** - Kræver tilslutning af hardware
- **RoboLibrary:Reader** - Kræver tilslutning af hardware
- **RoboLibrary:Writer** - Kræver tilslutning af hardware

Derudover er der også ekskluderet diverse metoder og få klasser der kun har til formål at debugge PC Applikation.

Bilag

Bilag 4 Code Coverage
Bilag 11 Kildekode
Bilag 12 Kodedokumentation
Bilag 22 Sætningsliste
Bilag 23 Tekniske specifikationer UR10
Bilag 24 Test Resultater
Bilag 28 User Manual UR10

Litteratur

[1] Wikipedia. Laplacian smoothing. https://en.wikipedia.org/wiki/Laplacian_smoothing. Senest besøgt den 5. december 2016.