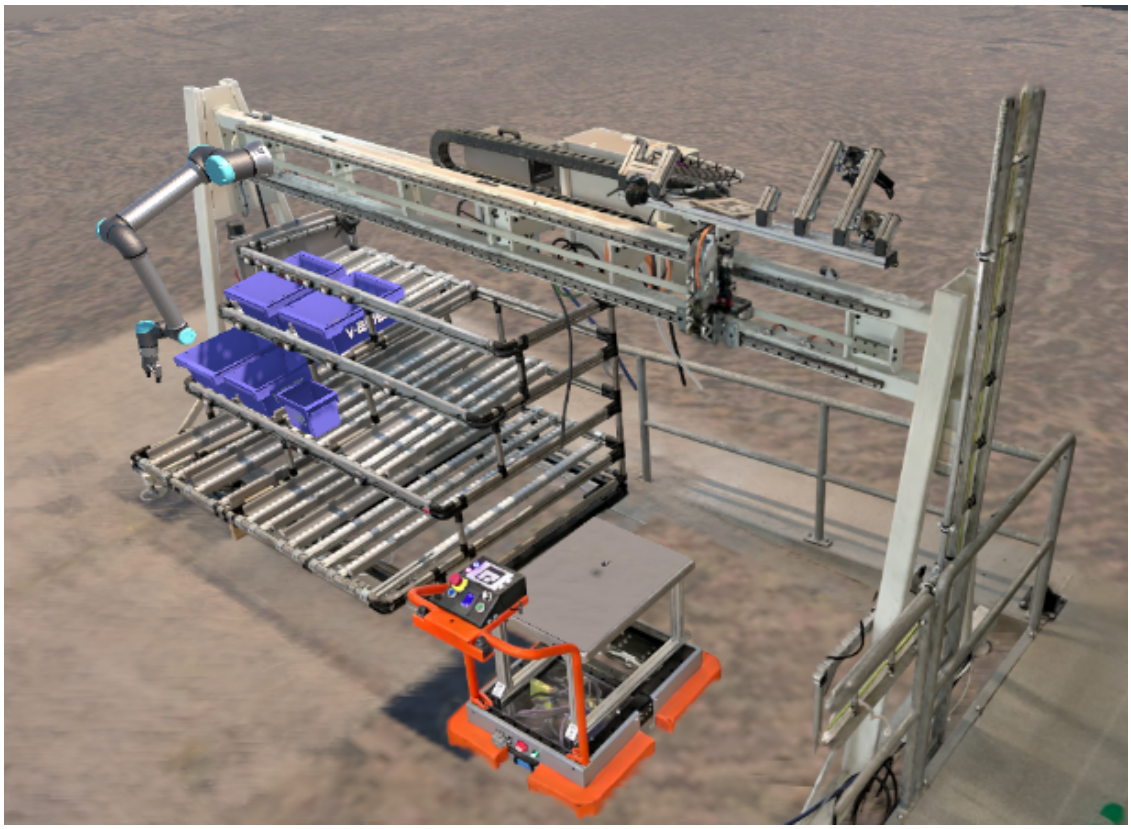
CHALMERS



Säker människa-robot-interaktion i fotorealistisk simuleringsmiljö

Utveckling av en digital tvilling med 3D Gaussian Splatting för simulering av kollaborativ robotik i Unity

Kandidatarbete inom Elektroteknik

WILHELM ALNERVIK, TILDA BENGTSSON, ERIC BERG, SOFIA FORSBERG, SIMON KRISTENSSON, CALLE WALLERSTEDT

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2026
www.chalmers.se

Kandidatarbete 2026

Säker människa-robot-interaktion i fotorealistisk simuleringsmiljö

Utveckling av en digital tvilling med 3D Gaussian Splatting
för simulering av kollaborativ robotik i Unity

WILHELM ALNERVIK
TILDA BENGTSSON
ERIC BERG
SOFIA FORSBERG
SIMON KRISTENSSON
CALLE WALLERSTEDT

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg 2026

Säker människa-robot-interaktion i fotorealistisk simuleringsmiljö
Utveckling av en digital tvilling med 3D Gaussian Splatting för simulering av kollaborativ robotik i Unity
WILHELM ALNERVIK, TILDA BENGTSSON, ERIC BERG, SOFIA FORSBERG,
SIMON KRISTENSSON, CALLE WALLERSTEDT

Handledare: Knut Åkesson, Elektroteknik
Examinator: Martin Fabian, Elektroteknik

Kandidatarbete 2026
Institutionen för Elektroteknik
EENX16-VT26-16B
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Robotarm som arbetar i 3D Gaussian Splatting kittningsstation.

Skriven i LaTeX
Göteborg 2026

Abstract

The development of automated industrial environments increases the need for a safe and efficient transition toward fully automated factories where humans and robots can collaborate. One effective approach for enabling this is the creation of a digital twin that can replicate reality within a simulated environment for simulation, testing, and validation before implementation in real production environments.

This bachelor's thesis investigates how *3D Gaussian Splatting* (3DGS) can be used to create a realistic digital twin of an industrial robot cell. The model is based on the RITA-cell, a kitting environment containing a collaborative robot arm (cobot), and is implemented in the Unity game engine for simulation and visualization.

The project includes the development of a 3DGS-based simulation model integrated into Unity. In addition, an algorithm for generating synthetic training data and a YOLO-based segmentation model were implemented for identifying components within the RITA-cell. Motion planning for the cobot was developed through communication between Unity and MoveIt using ROS 2, and a simplified sensor fusion solution was implemented to coordinate camera data in a shared coordinate system. These subsystems were primarily developed and tested as separate parts of the project. Furthermore, VR was chosen as the interaction method for visualization and manipulation of the simulated environment.

The results show that 3DGS can be used in an efficient and practical way to create visually realistic digital environments, although with certain limitations related to mesh geometry, which resulted in challenges regarding collision handling and physical simulation. The developed environment showed potential as a platform for robotic application development, but the final demonstration should mainly be viewed as a proof-of-concept since motion planning and sensor fusion were not fully integrated into the final simulation environment.

The conclusion is that 3DGS is well suited for creating visually realistic digital twins of industrial environments, but that the lack of straightforward mesh geometry integration makes it less suitable as a complete basis for physical simulation. The developed environment demonstrates potential as a platform for the development of robotic applications, provided that the visual model is combined with separate collision geometry and more robust integration of perception, sensor fusion, and robot control.

Keywords: digital twin, 3D Gaussian Splatting, Unity, segmentation

Sammandrag

Utvecklingen av automatiserade industrimiljöer ökar behovet av en säker och effektiv övergång till fullt automatiserade fabriker där människor och robotar kan samarbeta. En effektiv metod för detta är att skapa en digital tvilling som kan spegla verkligheten i en simulerad miljö för simulering, testning och validering innan implementation i den praktiska produktionen.

Detta kandidatarbete undersöker hur *3D Gaussian Splatting* (3DGS) kan användas för att skapa en realistisk digital tvilling av en industriell robotcell. Modellen baseras på RITA-cellen, en kittningsmiljö med en kollaborativ robotarm (cobot), och implementeras i grafikmotorn Unity för simulering och visualisering.

Arbetet omfattar framtagning av en 3DGS-baserad simuleringsmodell som integreras i Unity. Vidare implementerades en algoritm för framtagning av syntetisk träningsdata och en segmenteringsmodell baserad på YOLO för identifiering av komponenter i RITA-cellen. Cobotens banplanering utvecklades genom ett flöde mellan Unity och MoveIt med hjälp av ROS 2. För att samordna kameradata i ett gemensamt koordinatsystem implementerades även en förenklad sensorfusionsfunktion. Dessa delsystem utvecklades och testades främst som separata delar av projektet. Med simuleringsmodellen krävdes även en interaktionsmetod för visualisering och manipulering av systemet, här valdes därför VR som interaktionsmetod.

Resultatet visar att 3DGS på ett enkelt och effektivt sätt kan skapa visuellt realistiska digitala miljöer, men med vissa begränsningar i form av meshgeometri som ledde till utmaningar gällande kollisionshantering och fysisk simulering. Miljön visade potential som plattform för utveckling av robotapplikationer, men slutdemonstrationen bör främst ses som ett proof-of-concept eftersom styrning och sensorfusion inte integrerades fullt ut i den slutliga simuleringsmiljön.

Slutsatsen är att 3DGS är lämpad för att skapa visuellt realistiska digitala tvillingar av industriella miljöer, men att den saknar enkel integrering av meshgeometri vilket gör den mindre lämpad som ensam grund för fysisk simulering. Miljön som skapades visar potential som plattform för utveckling av robotapplikationer, förutsatt att den visuella modellen kompletteras med separat kollisionsgeometri och mer robust integration av perception, sensorfusion och robotstyrning.

Nyckelord: digital tvilling, 3D Gaussian Splatting, Unity, segmentering

Förord

Detta kandidatarbete har genomförts under våren 2026 vid institutionen för elektroteknik på Chalmers tekniska högskola. Arbetet har utförts inom området digitala tvillingar, robotik och perception, med fokus på hur realistiska simuleringsmiljöer kan användas vid utveckling av industriella robotapplikationer.

Vi vill rikta ett stort tack till vår handledare Knut Åkesson för vägledning, återkoppling och stöd under projektets gång. Vi vill även tacka vår examinator Martin Fabian.

Ett särskilt tack riktas till Volvo AB och Atieh Hanna för möjligheten att arbeta med RITA-cellen och för den hjälp vi fått kopplad till den industriella miljön. Tillgången till cellen har varit en viktig förutsättning för projektets genomförande.

Wilhelm Alnervik, Tilda Bengtsson, Eric Berg, Sofia Forsberg, Simon Kristensson, Calle Wallerstedt, Göteborg, maj 2026

Tillkännagivande om AI användning

I detta arbete har AI-verktyg använts för programmering, felsökning, rådgivning, formatering av referenser och för att ge feedback på text. Allt innehåll har granskats av författarna, som också ansvarar för det.

Akronymer

Nedan följer en lista över akronymer som har använts i denna rapport, listade i alfabetisk ordning:

3DGS	3D Gaussian Splatting
AMR	Autonom Mobil Robot
CAD	Computer-Aided Design
CHOMP	Covariant Hamiltonian Optimization for Motion Planning
IK	Inverse Kinematics
LCC	Lixel Cyber Color
NeRF	Neural Radiance Fields
PLY	Polygon File Format
RGB	Red Green Blue
ROS	Robot Operating System
SAM	Segment Anything Model
STOMP	Stochastic Trajectory Optimization for Motion Planning
UE5	Unreal Engine 5
VR	Virtual Reality
YOLO	You Only Look Once

Figurer

Innehåll

1
Introduktion

Utvecklingen inom industriell automation har lett till att robotar och människor i allt större utsträckning arbetar tillsammans i gemensamma arbetsmiljöer. För att möjliggöra säkra och effektiva robotsystem krävs avancerade metoder för simulering, perception och banplanering. Samtidigt ökar behovet av realistiska digitala miljöer där nya system kan utvecklas, testas och valideras innan de implementeras i verklig produktion. Digitala tvillingar har därför blivit ett viktigt verktyg inom industrin. Genom att kombinera simulering, sensorinformation och visuella miljömodeller kan digitala tvillingar användas för att analysera och utveckla robotsystem på ett mer flexibelt och kostnadseffektivt sätt. I detta arbete undersöks hur tekniker som *3D Gaussian Splatting* (3DGS), segmentering, sensorfusion och robotstyrning kan integreras för att skapa en digital tvilling av en industriell robotcell i en digital miljö som användaren kan interagera med via *Virtual Reality* (VR).

1.1 Bakgrund

I en värld där tekniken ständigt utvecklas strävar många företag efter att hålla jämna steg med förändringarna för att förbli konkurrenskraftiga. En stor del av denna utveckling sker inom automatiserade industrimiljöer. För att möjliggöra en säker och effektiv övergång till en fullt automatiserad industri där människor och robotar kan samarbeta, är en effektiv metod för detta att skapa en digital tvilling. De används för att skapa virtuella representationer av produktionssystem och kan användas för att simulera, analysera och optimera innan förändringar eller förbättringar implementeras i verkligheten [1]. Digitala tvillingar fungerar som stöd för så kallade ”what-if”-analyser, vilket betyder att nya arbetsflöden, layouter eller robotrörelser kan testas utan att störa den verkliga produktionen [2].

Kollaborativa robotar blir allt vanligare i de moderna fabrikerna där människor och robotar delar arbetsyta. Här är centrala utmaningar säkerhet, interaktion och tillit för att få dessa system att fungera i praktiken [3]. Roboten behöver uppfatta människor, objekt och förändringar i arbetsområdet för att kunna agera på ett säkert sätt. Därför används ofta sensorbaserade metoder, såsom segmenteringsmodeller och sensorfusion, för att upptäcka människor, hinder och potentiella risksituationer [4].

En central del i digitala tvillingar är även att minska sim-to-real gapet. Ofta finns det en skillnad i hur robotsystemet fungerar i simulering jämfört med verkligheten och detta kallas sim-to-real gap [5]. För att den digitala tvillingen ska kunna användas som en representation av det verkliga systemet är det avgörande att de liknar varandra och har samma funktionalitet. Detta eftersom mer realistiska simuleringsmiljöer kan förbättra möjligheterna att testa perception, banplanering och säkerhetsfunktioner, vilket kan minska både risker och kostnader vid utveckling av robotsystem [1].

De senaste åren har en ny teknik kallad 3DGS [6] blivit relevant inom visualisering och digitala tvillingar. Metoden möjliggör fotorealistisk rekonstruktion utifrån bild- eller videodata och kan skapa visuellt realistiska modeller med realtidsrendering och med begränsad manuell modellering.

AB Volvo CampX arbetar i nuläge med en kollaborativ robotcell RITA-cellen, där RITA står för *Robot In The Air*. RITA-cellen representerar en industriell kittningsmiljö där statisk cellstruktur, komponenthantering, perception, banplanering och mänsklig närvaro behöver samverka. Cellen fungerar därför som ett relevant testfall för en digital tvilling. Projektet ämnar därför att undersöka hur 3DGS, segmentering, sensorfusion och banplanering kan kombineras för att skapa en digital tvilling över RITA-cellen som kan visualisera miljön och användas för utveckling av säkrare människa-robot-interaktion.

1.2 Syfte

Syftet med projektet är att utveckla och utvärdera huruvida en digital tvilling av RITA-cellen kan användas för att minska sim-to-real gapet. Detta genomförs i Unity genom 3DGS tillsammans med segmentering, sensorfusion och cobotstyrning för att realistiskt återskapa och simulera RITA-cellen i en industriell miljö.

1.3 Frågeställning

- Hur väl kan 3D Gaussian Splatting användas för att skapa realistiska digitala tvillingar av industriella miljöer?
- Hur användbara är dessa digitala tvillingar för simulering och utveckling av robotapplikationer?

1.4 Avgränsningar

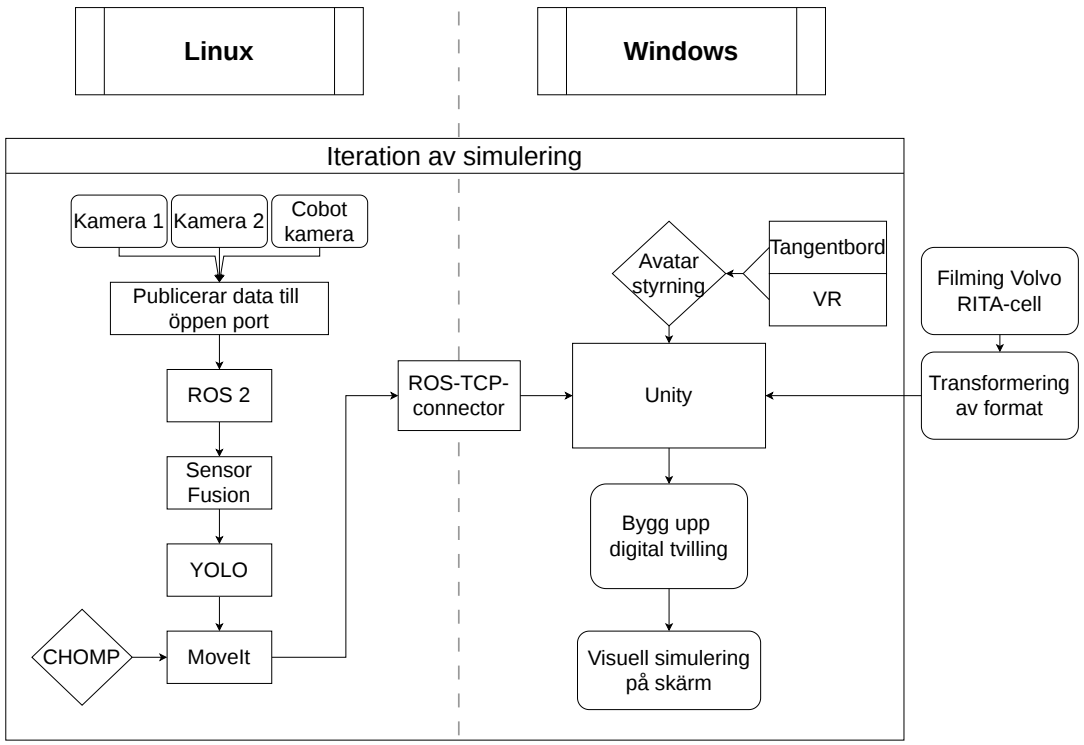
Projektet simulerar realistiska digitala tvillingar av industriella miljöer. Den digitala tvillingen är baserad på en verklig RITA-cell hos Volvo AB. Simuleringarna är avgränsade till att utföras i grafikmotorn Unity enligt handledarens instruktioner. Då arbetets fokus har varit på integrering så har arbetet utförts med redan befintlig kod från öppna källkodsprojekt.

Arbetet fokuserar på att skapa en simuleringsmiljö för att implementera industrirobotar. Därav har robotarmens funktionalitet avgränsats till en enkel banplaneringsalgoritm mellan förutbestämda punkter och använder inget fysiksimulerat gripande. I stället komponenten hårdkodats fast i gripklon vid upplockning.

2
Ansats

Projektet omfattar några olika tekniska områden, bland annat simulering, perception, sensorfusion, robotstyrning och VR. För att hantera omfattningen delades arbetet upp i separata delproblem med egna arbetsprocesser och tekniska lösningar som först utvecklas och utvärderas individuellt för att sedan integreras till en gemensam digital tvilling av RITA-cellen. I huvudsak användes grafikmotorn Unity som simulerings- och visualiseringsmiljö, dit projektets alla olika delmoment sammanfördes. Detta kapitel beskriver projektets övergripande ansats, nedbrytningen av delproblem som kan ses i figur 2.1, samt de tekniska val som gjorts under arbetets gång. Den färdiga digitala tvillingen med de olika implementerade delmomenten kan ses i figur 2.2.

2.1 Nedbrytning



Figur 2.1: Flödesschema över arbetsprocessen uppdelat i vilka delar som gjorts på Linux respektive Windows.

Figur 2.2: Slutligt resultat av den digitala tvillingen, med 3DGS-modell, rörlig cobot, blå lådor med komponenter och *Autonom Mobil Robot* (AMR) som hämtar kittade komponenter.

2.1.1 Simuleringsmodell

Det första delproblemet var att skapa en realistisk digital modell av den industriella miljön. Denna modell behövdes som grund för simulering, visualisering och vidare utveckling av projektets övriga delar.

Miljön modellerades med 3DGS, eftersom metoden möjliggör snabb framtagning av visuellt realistiska modeller från videodata och samtidigt kan användas i en interaktiv miljö. Indata till denna del var videomaterial från den verkliga miljön. Utdata var en digital modell importerad till Unity, som användes som visuell grund för resten av systemet.

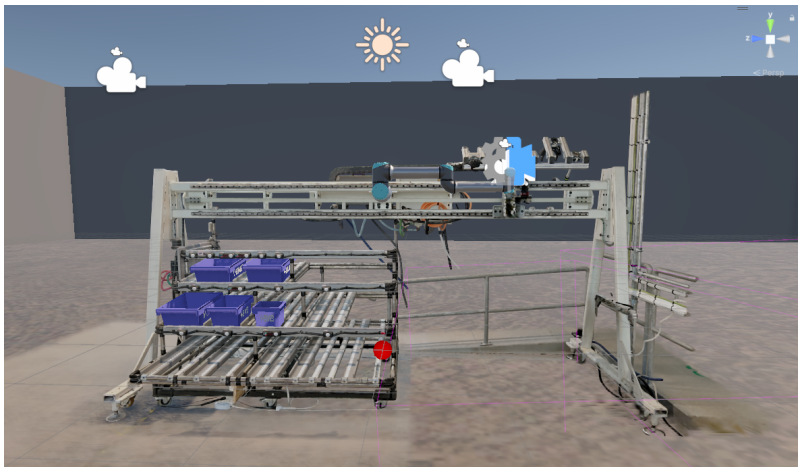
2.1.2 Styrning av Cobot

Styrningen av coboten var ett av delproblemen i arbetet. För att coboten skulle kunna röra sig i simuleringen krävdes ett styrningsflöde där en målposition kunde anges, exempelvis positionen för en låda där en viss komponent befinner sig. Därefter behövde coboten kunna beräkna och utföra en rörelse till den önskade positionen.

Ansatsen var att använda befintliga verktyg för robotstyrning och banplanering istället för att utveckla en egen banplaneringsalgoritm. Unity användes som simulerings- och visualiseringsmiljö, *Robot Operating System* (ROS) 2 som kommunikationsramverk och MoveIt som banplanerare. Delproblemet handlade därför om att sammankoppla dessa delar så att en målposition i Unity kunde skickas till ROS 2, beräknas i MoveIt och sedan skickas tillbaka som en ledtrajektoria som den simulerade UR10e-roboten kunde följa.

2.1.3 Segmentering

Det tredje delproblemet var att identifiera och avgränsa enskilda komponenter i arbetsmiljön. Detta behövdes eftersom coboten genom kameran på gripklon, se figur 2.3, måste kunna skilja mellan olika objekt och avgöra deras utbredning i bilden.



Figur 2.3: Figuren visar RITA-cellen från framsidan. Symboler för två takkameror kan observeras i luften ovanför RITA-cellen samt ytterligare en kamerasymbol är placerad på cobotens gripklo.

För detta användes instanssegmentering med en *You Only Look Once* (YOLO)-baserad modell. Metoden valdes eftersom den ger pixelvisa masker och samtidigt har potential att köras snabbt nog för praktisk användning. Indata till denna del var bilder av komponenter samt syntetiskt genererad träningsdata. Utdata var segmenteringsmasker som kunde användas för vidare analys av komponenternas position och form.

2.1.4 Sensorfusion

För att projektets olika delsystem ska kunna fungera ihop behövs en gemensam hantering av informationsflödet. Det är här delproblemet sensorfusion används för att sammanställa och koordinera data från tre kameror, se figur 2.3 så att den digitala miljön bygger på ett enhetligt koordinatsystem. Detta gör att alla funktioner byggs upp från samma system. Vid segmentering behöver flera sensorers data matchas med varandra för att undvika att skapa dubbla segmenteringar eller liknande objekt som identifieras i arbetsområdet. Genom att kombinera information på detta sätt kan man få en mer robust och tillförlitlig uppfattning av omgivningen. Detta bidrar i sin tur till ökad redundans, vilket innebär att systemet kan fungera även om enskilda sensorer är påverkade av störningar eller tillfälliga fel.

2.1.5 Interaktion med simuleringsmiljön

Det sista delproblemet innefattar att VR och en mänsklig avatar implementeras i Unity för att möjliggöra en immersiv och realistisk interaktion med den digitala tvil-

lingen. VR valdes eftersom det ger användaren en realistisk och intuitiv upplevelse av den renderade miljön jämfört med en skärm. För att interagera med simuleringen användes VR-headsetet tillsammans med handkontrollerna och kopplades till avataren. Genom detta synkroniserades användarens rörelser i verkligheten till avataren i simuleringen. Handkontrollerna möjliggör interaktion med avatarens händer till exempel upplockning av objekt.

2.2 Grafikmotor

För rendering och simulering av den digitala tvillingen krävs en grafikmotor som kan simulera verklighetstrogen fysik och rendera komplexa miljöer i realtid. Vidare krävs robust integrationsstöd med externa komponenter och goda modifieringsmöjligheter till de andra komponenterna i projektet såsom segmenteringen, sensorfusion och ROS 2. För att säkerställa verklighetstrogen VR-upplevelse samt enkel implementering och användning krävs ett grundligt stöd för VR. Eftersom ett av målen med denna rapport är en utvärdering av hur väl en industriell miljö kan simuleras med 3DGS ställs krav på att grafikmotorn stödjer 3DGS-rendering. Utöver detta är det meriterande om programmet har ett användarvänligt gränssnitt för att effektivisera arbetet. Som följd av dessa behov står valet av grafikmotor mellan Unity och *Unreal Engine 5* (UE5), då båda uppfyller de identifierade kraven och är de två största kostnadsfria grafikmotorerna i industrin för studenter [7]. Eftersom varken Unity eller UE5 har inbyggt stöd eller officiella plugins för 3DGS-rendering krävs plugins från tredjepartsutvecklare.

Utifrån de identifierade behoven jämförs Unity med UE5 för att bedöma den mest lämpliga grafikmotorn för projektet. Båda motorerna har omfattande funktionalitet för realistisk rendering i realtid av komplexa miljöer och har plugins för 3DGS rendering från tredjepartsutvecklare. UE5 erbjuder dock mer avancerade renderingstekniker som presterar bättre för stora komplexa miljöer, men kommer på bekostnad av högre hårdvarukrav. Unity har däremot lägre hårdvarukrav och presterar bättre i mindre miljöer men erbjuder inte samma nivå av grafisk realism.

För att effektivt utföra projektet behöver grafikmotorn vara användarvänligt med enkel inlärning. Båda motorerna har grundligt stöd och enkel implementering av VR, samt goda modifieringsmöjligheter med externa komponenter. Unity erbjuder ett användarvänligt gränssnitt med en låg inlärningskurva och huvudsakligen använder C#, i kontrast till UE5 med brant inlärningskurva och som använder C++ [7][8].

Även om UE5 erbjuder bättre grafik med mer avancerade renderingstekniker och klarar större miljöer så valdes Unity för dess lägre hårdvarukrav och mindre inlärningskurva. Parallellt använder kandidatarbetets parallellgrupp UE5 för att utvärdera hur väl 3DGS kan användas för att simulera industriella miljöer.

3

Simuleringsmodell

Syftet med modellbygget var att skapa en visuell representation av RITA-cellen som kunde användas som en del av en digital tvilling. Fokus låg på att återskapa miljön med tillräckligt hög visuell realism för att modellen skulle kunna integreras i en interaktiv simuleringsmiljö. Detta kapitel behandlar val av metod för digital rekonstruktion av cellen, arbetsgången för modellbygget enligt flödesschemat i figur 3.2, samt en utvärdering av hur väl modellen fungerar i simuleringsmiljön.

3.1 Jämförelse mellan fotogrammetri, NeRF och Gaussian Splatting

Det finns flera olika metoder för att återskapa verkliga miljöer digitalt. En traditionell metod är fotogrammetri [9], där många bilder används för att rekonstruera en polygonbaserad 3D-modell. Fotogrammetri kan ge hög geometrisk detalj och redigerbara meshmodeller, men arbetsflödet är ofta tidskrävande och kräver omfattande bildinsamling samt efterbearbetning.

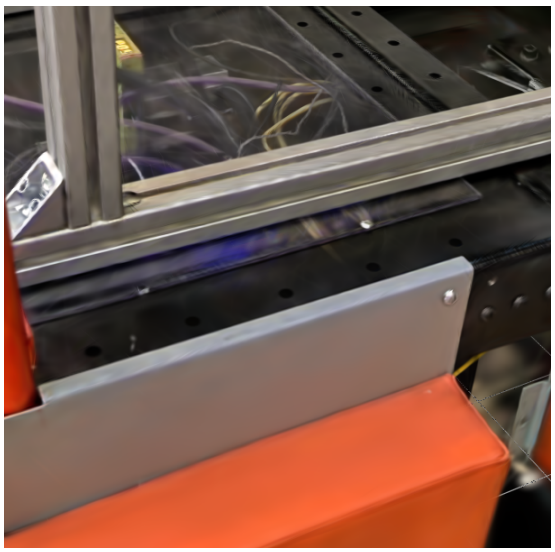
En annan metod är *Neural Radiance Fields* (NeRF) [10], där ett neuralt nätverk tränas för att representera hur ljus och färg varierar i en scen. NeRF kan ge mycket hög visuell kvalitet, men metoden är beräkningstung och mindre lämplig för realtidsrendering i interaktiva miljöer.

3DGS [6] kan ses som ett alternativ vilket kombinerar vissa fördelar från båda angreppssätt. I stället för att representera en scen med polygoner eller enbart med ett neuralt fält representeras den med ett stort antal tredimensionella gaussiska primitiver, se figur 3.1. Varje sådan gaussisk komponent har en position, orientering, storlek och färginformation. Tillsammans bildar de en visuell approximation av scenen som kan renderas effektivt och med hög realism.

Den huvudsakliga fördelen med 3DGS i detta projekt var kombinationen av visuell kvalitet och relativt snabb rendering. Metoden är särskilt intressant när målet är att återskapa en verklig miljö på ett sätt som upplevs trovärdigt för en mänsklig observatör, även om representationen inte i första hand är avsedd för exakt geometri eller fysikalisk simulering.

En ytterligare fördel med 3DGS i projektet var att modellen kunde placeras i Unity

med bibehållen skala, förutsatt att export och import genomfördes utan omskalning. Detta underlättade integreringen av modellen i den digitala tvillingen.

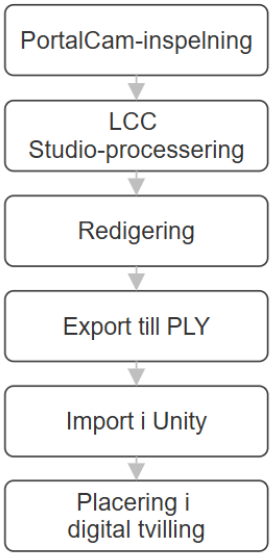


(a) Rendering av 3DGS-modell av AMR.

(b) Markerade ovala gaussiska primitiver synliga i 3DGS-modellen.

Figur 3.1: Visualisering av tredimensionella gaussiska primitiver i en 3DGS-modell med redigeringsverktyget SuperSplat. Figur (a) visar en 3DGS-modell och figur (b) markerar gaussians som modellen består av.

3.2 Beskrivning av metod och arbetsflöde



Figur 3.2: Flödesschema över 3DGS-modellbygge. Flödesschemat visar stegen från filmning till rendering i Unity.

RITA-cellen är en robotcell uppbyggd kring en horisontell skena i cellens överdel, där coboten är monterad. Coboten kan förflyttas horisontellt längs cellen och plocka komponenter från utbytbara lådor placerade under delar av skenan. Cellens konstruktion består till stor del av öppna ramverk, vilket ger den en skelettliknande geometri snarare än en kompakt volym. Cellen är 4.6 meter lång, 1.7 meter bred och 2.1 meter hög. Ytorna är av metall men inte av reflektiv karaktär och hela cellen är jämt belyst av takbelysning, vilket minimerar ljusvariationer.

Figur 3.3: Bearbetad 3DGS-modell av RITA-cellen i *Lixel Cyber Color*(LCC)-studio.

3.2.1 Utrustning och datainsamling

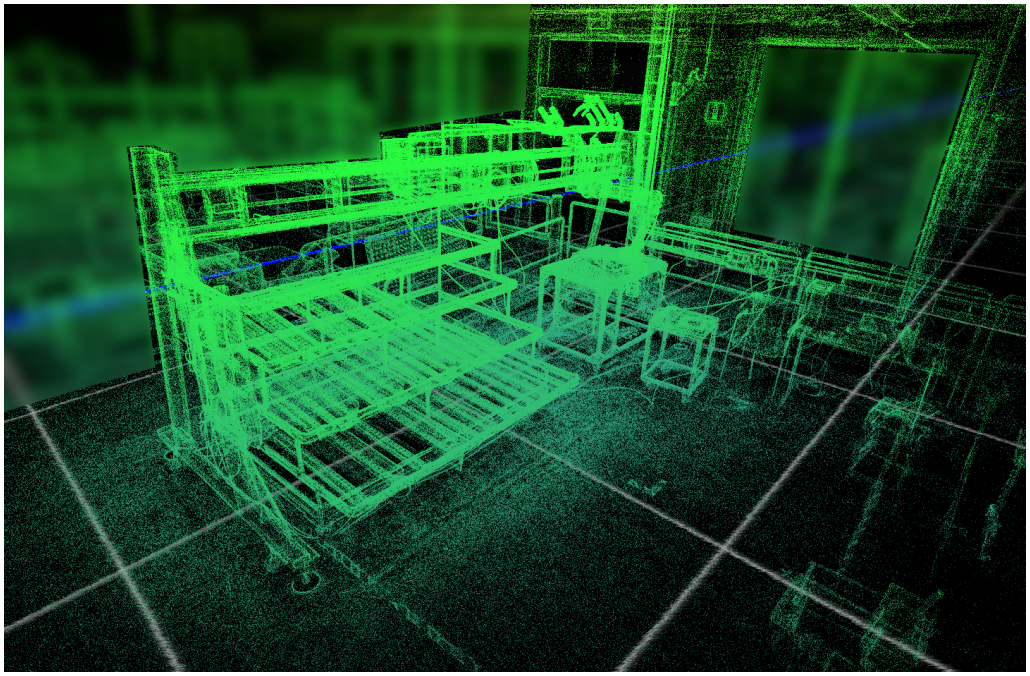
3DGS-modeller kan genereras från bildmaterial extraherade från video, där bildsekvensen bearbetas och återskapar scenen med gaussians. I detta projekt användes Xgrids PortalCam [11], som är designad för att bygga 3DGS-modeller, har 512 GB lokalt minne och ca 1 timmes batteritid. Kameran kommer med 2 utbytbara batterier. Arbetsflödet för inspelning med PortalCam genomfördes enligt följande steg:

1. Kameran anslöts till den tillhörande mobilapplikationen.
2. Inspelningen startades i mobilapplikationen.
3. Miljön filmades genom att kameran fördes genom cellen i låg gånghastighet.
4. Inspelningens täckning och kvalitet övervakades live i applikationen, där ett punktmoln visualiserades.
5. Inspelning avslutades i mobilapplikationen och datan sparades lokalt på kamerans hårddisk.

Inför inspelningen av RITA-cellen utfördes testinspelningar av PSL-labbet. Syftet var att undersöka batteritid, bästa kamerarörelse, gånghastighet och hur olika in-

spelningsstrategier påverkade resultatet. Testerna visade att bästa resultat genererades vid låg gånghastighet, där många vinklar av samma objekt fångades från både låga och höga riktningar med högt överlapp mellan vyer.

Kameran behövde hållas upprätt hela tiden utan att vinklas kraftigt. Eftersom kameran har flera linser kunde en bred vy fångas trots detta, men viss svårighet att fånga golvytor upptäcktes då ingen lins var riktad nedåt. Testerna visade även att reflektiva och genomskinliga ytor, exempelvis blank metall och glas, gav sämre rekonstruktionskvalitet. Rörliga personer i scenen lämnade artefakter i den slutliga modellen. Vid inspelning av RITA-cellen minimerades därför rörelse framför kameran. Inspelningen pågick under 18 minuter och en stege användes för att fånga cellens övre delar samt lådpositionerna ovanifrån.



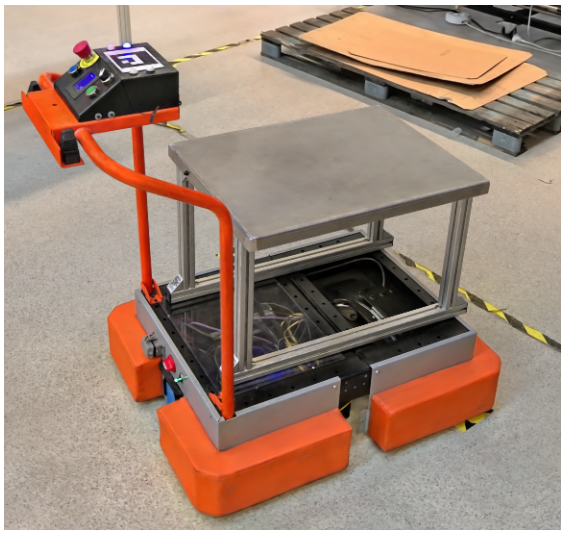
Figur 3.4: Punktmoln av RITA-cellen i *Lixel Cyber Color* (LCC) Studio efter bearbetning av insamlad data.

	RITA-cell	**AMR**
Inspelningstid	18 minuter	3 minuter
Bearbetningstid	6 timmar 51 minuter	24 minuter
Filformat	PLY	PLY
Filstorlek	191 649 KB	10 753 KB
Antal splats	2.88 miljoner	0.162 miljoner

Tabell 3.1: Detaljer om bearbetning för 3DGS-modeller av RITA-cellen och AMR.

3.2.2 Bearbetning och redigering av modellen

Efter datainsamlingen bearbetades inspelningen i LCC-studio, vilket är den tillhörande mjukvaran för PortalCam. Bearbetningen resulterade i en 3DGS-scen, se figur 3.3. Modellen kunde därefter exporteras i flera format, se appendix A.2 och A.3, däribland LCC, *Polygon File Format* (PLY), punktmoln, se figur 3.4, samt *Wavefront Object* (OBJ) för en enkel mesh. Bearbetningstiden berodde på inspelningens omfattning. För mindre modeller, exempelvis *Autonom Mobil Robot* (AMR) och coboten på låda, se figur 3.5, tog bearbetning runt 30 minuter när inspelningen var kring 3 minuter lång. För RITA-cellen, där inspelning pågått i 18 minuter, var bearbetningstiden 6 timmar och 51 minuter. Se tabell 3.1 för detaljer kring bearbetningen.



(a) Rendering av 3DGS modell av AMR i LCC Studio.

(b) Rendering av 3DGS modell av cobot på låda i LCC Studio.

Figur 3.5: 3DGS-modeller av andra robotar som ska arbeta i närheten av RITA-cellen.

Vid bearbetning i LCC Studio kunde parameterinställningar för rekonstruktionen göras, däribland huruvida den skulle göras snabbt eller långsamt, maximalt antal gaussians som kunde skapas, användning av VRAM och lågminnes rekonstruktion, se appendix figur A.1.

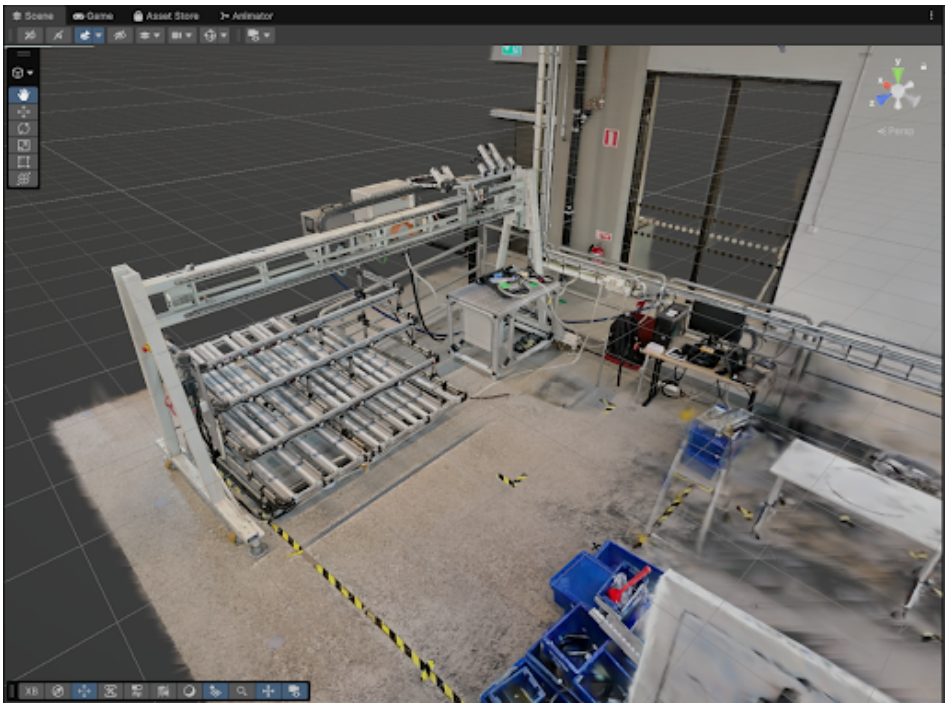
Efter bearbetning redigerades 3DGS-modeller direkt i LCCs mjukvara, där mindre modeller som cobot-bordet klipptes ut ur 3DGS-scenen före de exporterades. Modellen redigerades vidare på detaljnivå i SuperSplat [12], ett webbaserat verktyg för visualisering och redigering av 3DGS-modeller, där PLY-fil användes både vid import och export.

Till en början användes Xgrids egna LCC-format, eftersom formatet är anpassat för PortCam-systemet, var lättviktigt och hade effektiv rendering. Vid import till

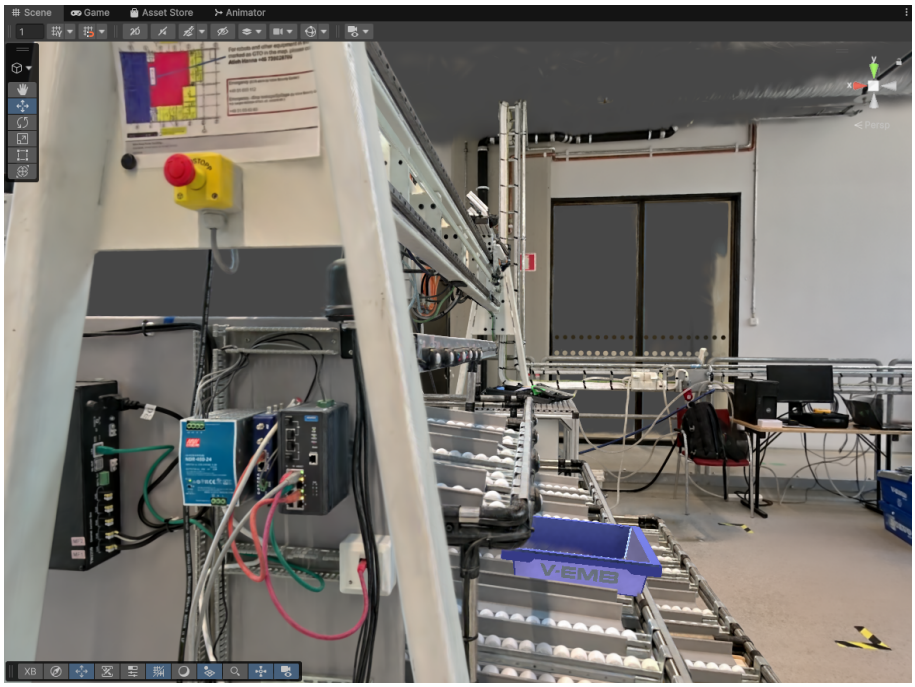
Unity uppstod dock problem med det tillhörande LCC-pluginet då det var svårt att integrera i projektets Unity-scen och var oklart om det gick att använda flera splatmodeller i samma scen. Problem uppstod även då endast en vinkel av miljön renderas samtidigt. Vilket orsakade att endast ett öga för VR renderades, det andra ögat och takkamerorna registrerade ingen splatt. Därför valdes istället PLY-formatet för import till Unity. PLY är det främst förekommande formatet för 3DGS-modeller och stöds av fler externa verktyg, bland annat redigeringsverktyget SuperSplat och Unity-pluginet Gaussian Splatting [13] som användes. Detta gav ett mer flexibelt format och arbetsflöde och möjliggjorde användning av flera 3DGS-modeller i samma scen. Vid första inspelningen av RITA-cellen fanns PLY-formatet ej bland exportvalen i LCC Studio för bearbetade 3DGS och en egen omformatering från LCC- till PLY-format gjordes därför. Konverteringen baserades på ett GitHub-repository som hanterade formatering mellan dessa format. LCC Studio utökade exportalternativen under projektets gång till att inkludera PLY, men på grund av problem i Unity scenen återgick arbetet till en backup med egenformaterad 3DGS.

3.2.3 Importering och användning i Unity

Den exporterade 3DGS-modellen från LCC Studio importerades till Unity med ett Gaussian Splatting-plugin [13] via PLY-formatet enligt instruktioner i pluginets GitHub-repository. Efter import stämde orienteringen inte alltid överens med koordinatsystemet i Unity-scenen utan behövde manuellt justeras. Modellens skala däremot bevarades så länge ingen omskalning skett under redigeringsprocessen.
I Unity kan även 3DGS-modeller maskeras med enkel geometri för att dölja delar av modellen, se figur 3.6 för den beskurna modellen i Unity.



Figur 3.6: Beskuren 3DGS av RITA-cellen i Unity. Artefakter från människor i rörelse kan ses till höger vid det vita bordet.



Figur 3.7: Detaljer på 3DGS-modellen där stationära objekt som sladdar och elektronik är synlig.

3DGS-modellen kunde kombineras med andra objekt som meshar och URDF-objekt i samma Unity-scen. Vid fler än en 3DGS i scenen skapar renderingsordningen vissa problem, eftersom den är statisk och inte flexibel utifrån vilket 3DGS-objekt som befinner sig närmast en kamera. Detta syns bland annat när AMR närmar sig RITA-cellen och bordet renderas framför cellen trots att den geografiskt befinner sig bakom, se figur 3.8. Eftersom 3DGS-modeller främst är visuella och inte innehåller användbar kollisionsgeometri kan de inte direkt användas för kollisionsdetektion i Unity. För fysisk interaktion och fysiksimulering krävs därför en separat kollisionsgeometri.

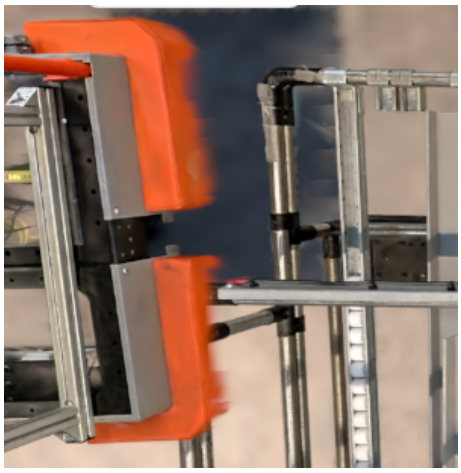
3.3 Utvärdering av modellbygge

Användningen av 3DGS gav en visuellt realistisk representation av RITA-cellen med relativt begränsad manuell modellering. Detta visar att modellen är väl lämpad för att snabbt skapa en visuell miljö av en befintlig robotcell, särskilt när syftet är visuellt snarare än geometrisk rekonstruktion.

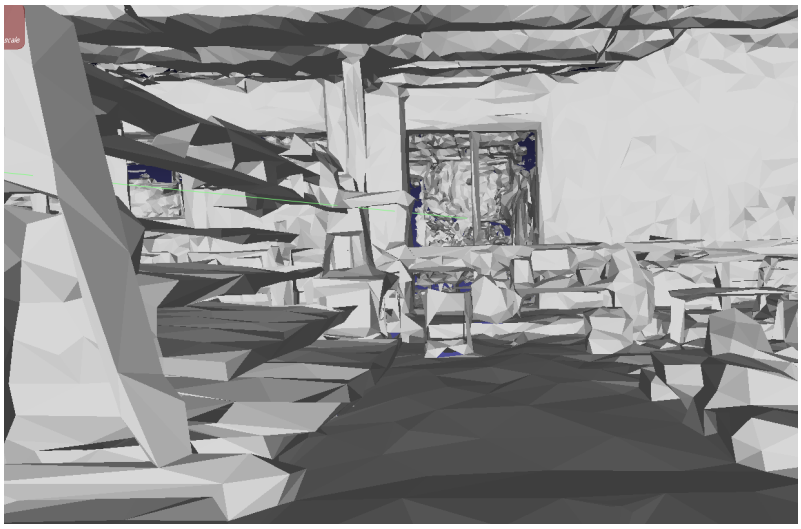
Modellen återgav cellens övergripande statiska struktur såsom skenor, lådplatser och omgivande utrustning som tillhörande elektronik och kablar, på ett tillräckligt sätt för att användas som ett visuellt objekt i Unity.

I detta projekt kunde bakgrunden av RITA-cellen inte inkluderas av sekretess, men vid framtida applikationer kan 3DGS-modeller användas för en trovärdig visuell bakgrund, där särskilt stationära objekt återskapas effektivt i modellen, se figur 3.7.

Arbetsflödet med PortalCam och LCC Studio krävde begränsad manuell modellering jämfört med att bygga modellen med exempelvis *Computer-Aided Design* (CAD). Däremot krävdes planering av inspelning, bearbetning, beskärning och anpassning för modellen i Unity.



Figur 3.8: Problematik med strikt renderingsordning. AMR med orange kant befinner sig geografiskt bakom RITA-cellens skenor, men renderas ovanpå.

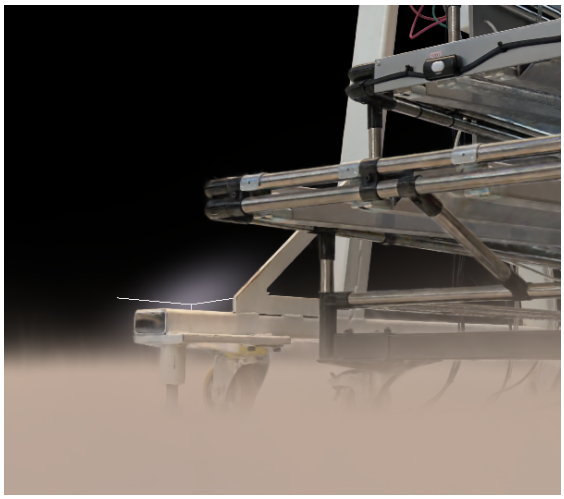


Figur 3.9: Exporterad mesh från LCC-studio efter bearbetning. Meshen är av enklare karaktär med begränsad detalj vilket inte räcker för kollisionsdetektering av RITA cellens individuella balkar när coboten arbetar.

Flera begränsningar identifierades, varav den huvudsakliga var avsaknaden av detaljerad mesh från LCC Studio. Eftersom 3DGS inte är en traditionell meshmodell saknar den meningsfull kollisionsgeometri. LCC Studio återskapade en mesh utifrån punktmolnet, men denna mesh var för grov för att kunna användas i Unity scenen, se figur 3.9 och är begränsad till användning för större, enformiga objekt med

grövre kollisionsdetektering. En framtida möjlighet är att generera en mesh utifrån punktmolnet PortalCam fångar med en tredjeparts mjukvara. Alternativt kan meshen genereras från 3DGS med exempelvis 3DGS-to-PC [14], för att skapa en mer detaljerad och användbar mesh för kollisionsdetektering.

Ytterligare begränsning var att vissa ytor blev visuellt oskarpa eller ”luddiga”, se figur 3.10, vilket var särskilt framträdande på ytor som var blanka, av tunn struktur och områden som filmats från för få vinklar, se figur 3.10. Resultatet visar att inspelningsstrategi har stor betydelse för modellens kvalitet. Stort överlapp av vyer och att täcka många vinklar bedömdes vara viktiga faktorer för en bra rekonstruktion.



(a) Visar rendering nära golvet på luddig golvytan i SuperSplat.

(b) Rendering av reflektiv yta i SuperSplat.

Figur 3.10: Svårigheter med vissa ytor uppstod vid modellbygget, särskilt vid golvytor och reflektiva ytor.

3.4 Sammanfattning av modellbygget

Resultatet påverkades tydligt av inspelningsstrategin. Långsam kamerarörelse, hög överlappning mellan vyer och täckning från flera vinklar var viktiga faktorer för god modellkvalitet. Begränsningar identifierades framför allt vid tunna strukturer, blanka ytor, golvytor och rörliga objekt i scenen.

3DGS-modellen bedöms vara väl lämpad som visuell bakgrund och referensmiljö i simuleringen. Däremot saknar modellen direkt användbar kollisionsgeometri och behöver därför kompletteras med CAD-modeller, meshobjekt eller förenklade kollisionsvolymer om fysisk interaktion eller kollisionsdetektion ska ingå i den digitala tvillingen.

4
Styrning av Cobot

I detta delproblem hanteras styrningen av coboten i RITA-cellen. Detta är en viktig del för att skapa en realistisk simuleringsmiljö. Arbetet fokuserar i stor del på att använda existerande tekniker. Det betyder att det är själva arbetsflödet som arbetet försöker lösa. Kapitlet tar upp valet av banplanering, hur styrningen implementerades samt utvärdering av metoden.

4.1 Bakgrund

För att coboten ska fungera på ett effektivt sätt behöver den styras och kommunicera med flera olika programmoduler. Detta kan göras med hjälp av ROS, ROS 2 [15]. ROS är ett mjukvaruramverk som fungerar som ett mellanlager mellan en dators operativsystem och robotens olika programmoduler, exempelvis sensordata, styrning, visualisering och banplanering.

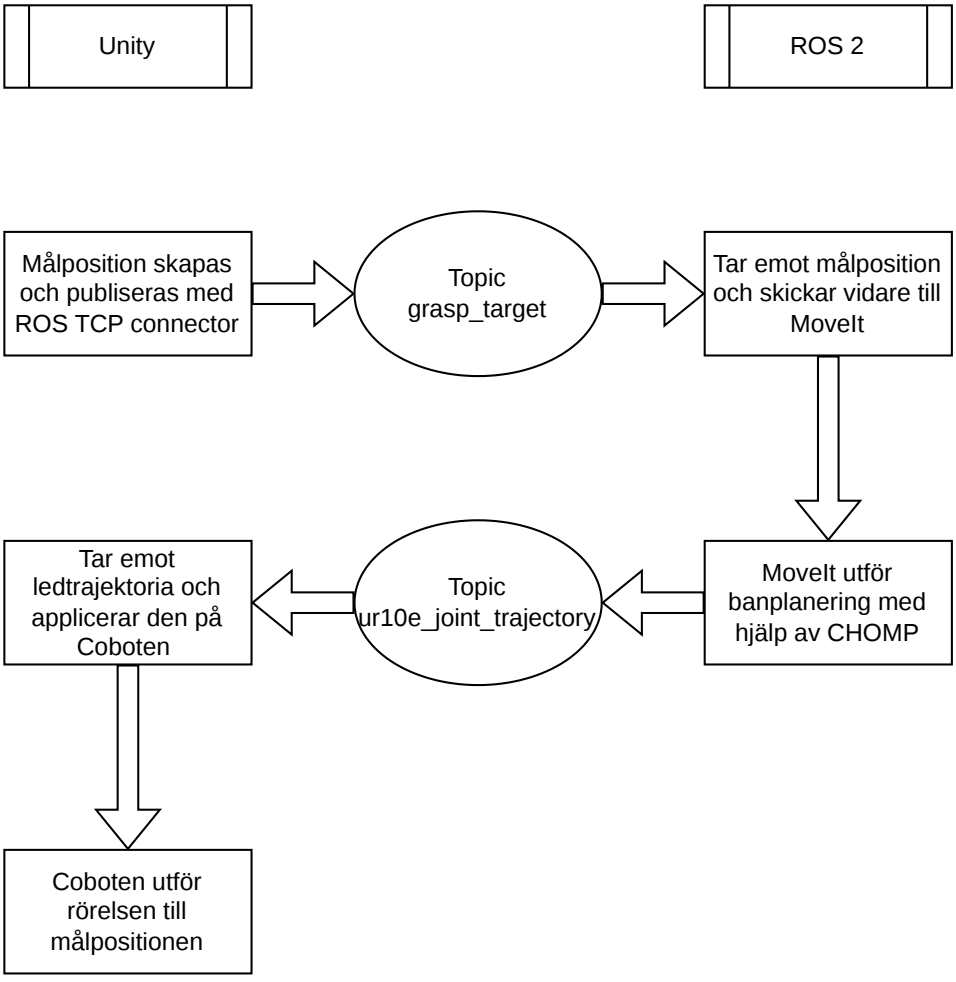
ROS 2 är en vidareutveckling av ROS och bygger på att funktionaliteten delas upp i separata noder [16]. Dessa noder kommunicerar med varandra via topics [17], där en nod kan publicera data och en annan nod kan prenumerera på samma data. ROS 2 är bättre anpassat än ROS 1 för distribuerade system, robust kommunikation, säkerhet och realtidsnära tillämpningar. I detta projekt används ROS 2 som kommunikationsramverk mellan Unity-simuleringen och robotens banplanering.

I detta arbete syftar styrning främst på banplanering och exekvering av rörelser i den simulerade robotmodellen. Låg-nivåreglering av motorer, exempelvis moment- eller hastighetsreglering, behandlas inte. Fokus ligger i stället på hur en målposition kan omvandlas till en planerad ledtrajektoria för UR10e-roboten.

För banplaneringen används MoveIt [18], vilket är ett planeringsramverk för robotarmar som används tillsammans med ROS 2. I stället för att manuellt beräkna varje ledvinkel kan en målposition skickas till MoveIt. MoveIt använder robotens modell, ledbegränsningar och information om omgivningen för att beräkna en möjlig trajektoria.

En trajektoria för en robotarm beskriver inte bara verktygets väg i rummet, utan består i praktiken av flera ledpositioner vid olika tidpunkter. För UR10e innebär detta att varje punkt i trajektorian innehåller önskade värden för robotens leder, vilket gör att Unity kan uppdatera robotmodellens ledvinklar stegvis.

För att detta ska fungera krävs att Unity och ROS 2 använder en gemensam tolkning av robotens positioner och leder. Robotens modell, lednamn och koordinatsystem måste därför stämma överens mellan Unity och MoveIt. Unity ansvarar därmed inte för själva banplaneringen, utan fungerar som simuleringsmiljö och gränssnitt, medan MoveIt ansvarar för att beräkna en möjlig rörelsebana för robotarmen. Det övergripande flödet från målposition i Unity till utförd rörelse visas i figur 4.1.



Figur 4.1: Översiktligt arbetsflöde för styrningen. En målposition skapas i Unity och skickas via ROS 2 till MoveIt, där en trajektoria beräknas. Den planerade trajektorian skickas sedan tillbaka till Unity och appliceras på den simulerade UR10e-modellen.

4.2 Val av banplaneringsalgoritm

I detta arbete valdes en redan existerande metod för banplanering och trajektorieoptimering i stället för att utveckla en egen algoritm. Detta eftersom projektets fokus ligger på att integrera Unity, ROS 2 och MoveIt för styrning av en simulerad UR10e-robot, snarare än på att utveckla en ny planeringsalgoritm.

Det finns flera färdiga metoder för trajektorieoptimering, exempelvis *Stochastic Trajectory Optimization for Motion Planning* (STOMP) [19] och *Covariant Hamiltonian Optimization for Motion Planning* (CHOMP) [20]. STOMP bygger på stokastisk trajektorieoptimering där slumpmässiga variationer av en initial trajektoria används för att hitta en bana med lägre kostnad. CHOMP är i stället en metod för trajektorieoptimering som iterativt förbättrar en initial bana med avseende på bland annat kollisionsfrihet och mjukhet i rörelsen. I detta projekt valdes CHOMP.

Valet baserades på att CHOMP bedömdes vara väl anpassad för projektets simulerade miljö. Eftersom RITA-cellen är relativt statisk kan hinder som golv, räls och övrig cellgeometri beskrivas i förväg i MoveIt. CHOMP kan då ta hänsyn till dessa kända hinder under optimeringen av trajektorian. Metoden är även lämplig när målet är att skapa jämna och realistiska rörelser, vilket är viktigt eftersom robotens rörelser visualiseras och utvärderas i Unity-simuleringen.

4.3 Beskrivning av arbetsflöde

Som tidigare nämnt är målet med arbetet att skapa en simuleringsmiljö som kan användas för att undersöka och minska delar av sim-to-real gapet. Unity används därför som simulerings- och visualiseringsmiljö för den digitala robotmodellen, medan ROS 2 och MoveIt används för kommunikation respektive banplanering. Det implementerade arbetsflödet illustreras i figur 4.1.

För att styrningen ska fungera behöver Unity kunna kommunicera med ROS 2 och MoveIt. Detta görs med hjälp av ROS-TCP Connector [21], vilket möjliggör att Unity kan publicera och prenumerera på topics i ett ROS 2-workspace på samma nätverk. ROS 2 fungerar därmed som kommunikationslager mellan Unity och banplaneringen, medan MoveIt tillsammans med CHOMP används för att beräkna robotens trajektorier.

I Unity används separata hjälpfunktioner för att hantera robotmodellen och dess rörelse. En del av implementationen ser till att den importerade UR10e-modellen är stabil i simuleringen genom att förankra robotens bas och anpassa relevanta fysikinställningar. Detta gör att robotmodellen inte påverkas felaktigt av Unitys fysiksimulering när scenen startas.

En annan del av implementationen används för att applicera den planerade rörelsen

på robotmodellen. När en ledtrajektoria tas emot från ROS 2 kopplas lednamnen i meddelandet till motsvarande leder i Unitys UR10e-modell. Därefter uppdateras robotens leder stegvis över tid enligt den planerade trajektorian.

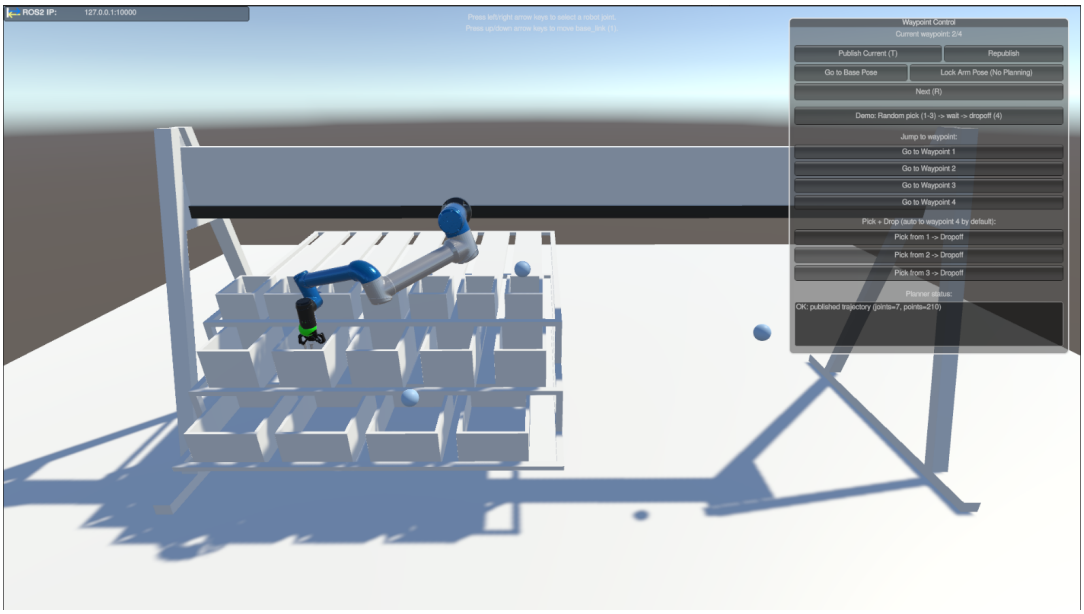
För att en trajektoria ska kunna beräknas skickas först en målposition från Unity till ROS 2. Målpositionen publiceras på ett topic som läses av en ROS 2-nod, vilken skickar positionen vidare till MoveIt. MoveIt använder därefter robotmodellen, ledbegränsningar och CHOMP för att försöka beräkna en giltig trajektoria till målpositionen. Om planeringen lyckas publiceras den färdiga ledtrajektorian tillbaka till Unity.

I det slutliga arbetsflödet används fördefinierade waypoints i Unity. Dessa waypoints representerar exempelvis positioner där komponenter kan hämtas samt en avlastningsposition där komponenterna ska lämnas. På så sätt kan roboten styras mellan relevanta punkter i simuleringsmiljön utan att varje målposition behöver anges manuellt.

Sammanfattningsvis gör arbetsflödet det möjligt att omvandla en förbestämd målposition i Unity till en planerad rörelse för den simulerade UR10e-roboten, vilket visas i figur 4.1. När en komponent efterfrågas skickas motsvarande målposition till ROS 2. Därefter beräknar MoveIt en trajektoria till målet och skickar resultatet tillbaka till Unity. Robotmodellen rör sig då enligt den planerade banan. Efter att roboten nått målpositionen kan en ny målposition skickas, exempelvis för avlastning, varpå samma planeringskedja används igen.

4.4 Utvärdering av styrning

När det kommer till resultatet för styrningen är det fungerande men begränsat. Det största och viktigaste positiva resultatet är att styrkedjan fungerar. Detta innebär att en målposition kan tas fram i Unity-simuleringsmiljön och därefter publiceras på rätt topic med hjälp av ROS-TCP Connector paketet för Unity. Målpositionen kan sedan läsas av i ROS 2 och skickas vidare till MoveIt. MoveIt använder informationen för att beräkna en trajektoria, som därefter publiceras tillbaka till Unity där den simulerade coboten utför rörelsen. Denna kedja motsvarar det övergripande flödet i figur 4.1.



Figur 4.2: Slutposition efter att coboten utfört den resulterande rörelsen framtagen i MoveIt med CHOMP. (Ingen 3DGS används i denna Unity-miljö.)

Generellt lyckas coboten även undvika kollision med kända objekt, såsom dess egna länkar, golvet samt delar av cellen under rörelsen till slutmålet likt bilden i 4.2. Detta visar att de objekt som är definierade i MoveIts planeringsmiljö kan tas hänsyn till vid banplaneringen. Däremot är inte alla objekt i simuleringsmiljön definierade i MoveIt. Exempelvis är lådorna där komponenterna är placerade inte inkluderade som kollisionsobjekt. Detta leder till att coboten i vissa fall försöker röra sig genom lådornas väggar. Lådorna definierades inte i MoveIt eftersom det påverkade banplaneringen negativt. När de var definierade ökade beräkningstiden betydligt, samtidigt som en stor del av planeringsförsöken resulterade i att ingen giltig trajektoria kunde beräknas. För att minska risken för kontakt med lådorna testades i stället en begränsning där coboten först endast fick röra sig ovanför lådan för att nå komponenten. Först när coboten befann sig ovanför lådan tilläts en nedåtgående rörelse in i lådan. Detta fungerade till viss del, men det valdes att inte undersöka metoden mer under arbetet. Utöver detta saknar även 3DGS objekt en användbar kollisions mesh vilket gör att även om beräkningstiden inte hade påverkats negativt av att ta hänsyn till cellens geometri skulle det inte kunna implementeras då det inte finns någon mesh att använda i MoveIt.

Cobotens rörelse skulle också kunna förbättras. I nuläget förekommer det ofta trajektorier som inte är optimerade. Det kan leda till att coboten utför stora cirkelrörelser eller omvägar för att nå målpositionen. Möjliga orsaker till detta kan vara att planeringsalgoritmens parametrar inte är optimerade för UR10e-robotarmen när den är vertikalt monterad på en linjär axel. Andra möjliga orsaker är ledbegränsningar, startpositionen eller målpositionens orientering.

Totalt sett innebär detta att resultatet för styrningen visar stora framgångar, men att det fortfarande finns problem som behöver åtgärdas. För detta arbete är det

viktigaste att kopplingen mellan simuleringsmiljön i Unity och MoveIt fungerar med hjälp av ROS 2, vilket den gör. Resultatet fungerar därför som en grund för fortsatt utveckling i projektet. Det som främst behöver förbättras är trajektoriernas effektivitet, planeringens robusthet och hur kollisionsobjekt i arbetsmiljön hanteras.

4.5 Sammanfattning

Syftet med denna del av arbetet var att skapa ett styrningsflöde där coboten i den simulerade miljön kunde styras genom beräkning av ledtrajektorier. Detta uppfylldes genom att använda Unity som simulerings- och visualiseringsmiljö, MoveIt tillsammans med CHOMP-algoritmen för att beräkna trajektorier, samt ROS 2 för att koppla samman de två delarna av systemet. Det övergripande flödet visas i figur 4.1.

Med detta skapades ett fungerande arbetsflöde där en målposition skapas i Unity och publiceras på ett topic i ROS 2. Denna målposition läses sedan av och skickas vidare till MoveIt, där CHOMP-algoritmen används för att beräkna en giltig trajektoria. Trajektorian publiceras sedan på ett separat topic, som i sin tur läses av i Unity, där coboten utför den önskade rörelsen.

Styrkedjan fungerar på ett tillförlitligt sätt och de planerade rörelserna utförs av coboten i simuleringsmiljön. Generellt undviks även kollision med de objekt som är definierade i MoveIts planeringsmiljö. Detta, tillsammans med fördefinierade mål för de olika lådorna, gör det möjligt att skapa en fungerande simulering av hur coboten kan användas i en verklig miljö.

Det finns dock begränsningar. Alla objekt i Unity-miljön är inte definierade som kollisionsobjekt i MoveIt, eftersom detta skapade stora problem för trajektorieberäkningen. Utöver detta blev vissa trajektorier ineffektiva, vilket ledde till onödiga omvägar för att nå målpositionen.

Delproblemet kan ses som uppfyllt eftersom en fungerande koppling mellan simuleringsmiljön och banplaneringen i MoveIt skapades med hjälp av ROS 2. Resultatet visar att en målposition kan efterfrågas i simuleringsmiljön och, efter beräkning i MoveIt, utföras av coboten. Lösningen är ännu inte en komplett autonom pick-and-place-styrning, men fungerar som en grund för fortsatt utveckling. Det främsta förbättringsområdet är trajektoriernas effektivitet, både för att skapa mer realistiska rörelser och för att ge en mer visuellt önskvärd simulering.

5
Segmentering

Detta kapitel beskriver hur segmentering användes för att identifiera komponenter i arbetsmiljön. Först motiveras valet av segmenteringsmodell, därefter beskrivs hur träningsdata skapades och hur modellen utvärderades på verkliga bilder. Kapitlet avslutas med metodens begränsningar och möjliga vidareutveckling.

5.1 Bakgrund

För att en robot ska kunna plocka och flytta komponenter i en kittingmiljö behöver den både kunna tolka sin omgivning och identifiera de objekt som ska hanteras. I RITA-cellen är stora delar av miljön statiska, vilket innebär att flera moment kan planeras i förväg. Exempelvis kan robotens förflyttning till en viss låda, till en placeringspunkt eller bort från en människa i arbetsområdet beskrivas med fördefinierade rörelser.

Det som inte kan hårdkodas på samma sätt är själva greppet av komponenterna. Komponenterna ligger inte alltid på samma plats eller i samma orientering i sina backar, vilket gör att roboten måste kunna avgöra var en viss komponent befinner sig och hur den är placerad. För att detta ska vara möjligt krävs en metod som ger mer information än enbart en grov lokalisering av objektet.

I detta arbete användes därför segmentering. Till skillnad från objektdetektion med begränsningsrutor ger segmentering en pixelvis mask av varje objekt i bilden. Denna information är mer användbar vid gripning, eftersom roboten då kan få en bättre uppskattning av komponentens utbredning, orientering och fria ytor.

5.2 Val av segmenteringsmodell

Det finns många modeller för bildsegmentering, men alla är inte lämpliga för detta användningsfall. De krav som ställdes på segmenteringslösningen var framför allt följande:

- låg inferenstid och möjlighet till realtidsnära användning,
- pixelvis segmentering snarare än enbart bounding boxes,
- möjlighet att specialisera modellen för ett begränsat antal komponenttyper.

Flera olika modeller och ansatser undersöktes. YOLO-familjen är framtagen för snabb objektdetektion och har vidareutvecklats i flera versioner med fokus på realtidsnära användning [22, 23]. I senare versioner och implementationer finns även stöd för instanssegmentering, där modellen inte bara predikterar klass och position utan även en mask för varje detekterat objekt.

Metas *Segment Anything Model* (SAM) är en generell segmenteringsmodell som kan skapa masker från olika typer av promptar och är tränad för att fungera på många typer av bilder [24]. Denna generalitet gjorde modellen intressant att testa, men också mindre lämpad som slutlig lösning i projektet eftersom användningsfallet var smalt och krävde låg inferenstid. Även FastSAM och SegFormer undersöktes. FastSAM är en snabbare variant inspirerad av SAM [25], medan SegFormer är en transformerbaserad modell för semantisk segmentering [26]. Dessa bedömdes dock inte ge rätt kombination av hastighet, enkel integration och praktisk användbarhet för projektets behov.

Valet föll därför på YOLO-baserad segmentering. Denna metod kräver egen träningsdata, men ger en modell som kan anpassas till projektets specifika komponenter och köras betydligt snabbare än mer generella segmenteringsmodeller.

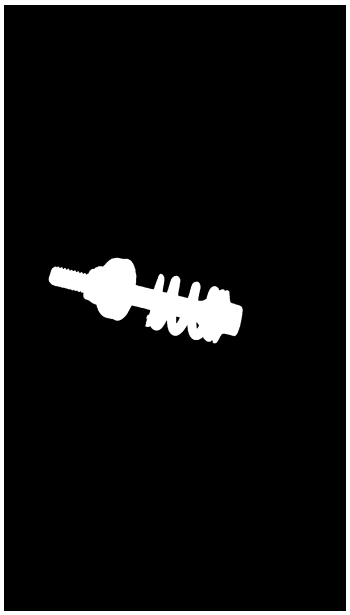
5.3 Framtagning av träningsdata för segmentering

För att träna en YOLO-segmenteringsmodell krävs träningsbilder av de komponenter som ska identifieras, samt motsvarande facit i form av segmenteringsmasker. En modell med god prestanda kräver vanligtvis ett stort antal annoterade bilder, vilket gör manuell annotering opraktisk i ett projekt av denna typ. Därför utvecklades i stället en pipeline för att skapa syntetisk träningsdata.

Utgångspunkten var att filma enskilda komponenter mot en ren bakgrund, exempelvis en greenscreen, från flera olika vinklar. I detta projekt användes en magentafärgad bakgrund för att göra komponenten lättare att separera från omgivningen.

Figur 5.1: Exempel på komponent filmad mot magentafärgad bakgrund. Den tydliga bakgrunden användes för att förenkla automatisk extraktion av komponenten.

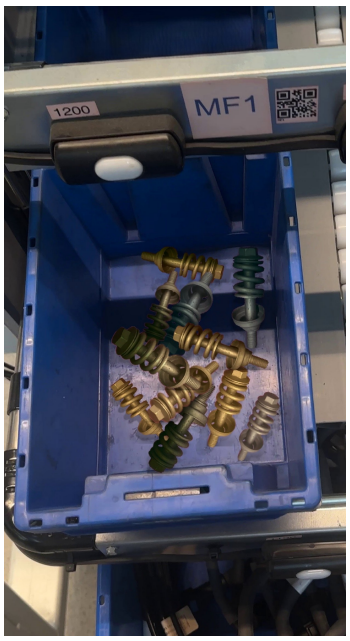
Med hjälp av en egenutvecklad algoritm kunde komponenten extraheras från videon. Detta gav både bildrutor med komponenten och en exakt mask som beskrev dess utbredning i bilden.



Figur 5.2: Automatiskt extraherad mask för komponenten. Masken används som facit vid generering av träningsdata.

De extraherade komponenterna placerades sedan automatiskt in i bilder av tomma lådor. För att öka variationen i träningsdatan och göra den mer representativ för verkliga förhållanden användes flera bildtransformationer, såsom skaländring, rotation, positionsvariation samt visuella effekter som skuggor och andra ljusrelaterade

justeringar. Hela denna process automatiserades i en algoritm som utvecklades för projektets specifika syfte.



Figur 5.3: Exempel på syntetiskt genererad träningsdata där extraherade komponenter placerats i en låda med varierad position, skala och rotation.

Resultatet blev ett träningsdataset bestående av syntetiskt genererade bilder av komponenter i backar, tillsammans med tillhörande segmenteringsmasker. Detta gjorde det möjligt att träna en modell utan att behöva manuellt annotera ett stort antal bilder.
Koden för datapipelinen, modellträningen, inferens och testning finns tillgänglig i ett GitHub-repository, se appendix.

5.4 Utvärdering av segmenteringsmetoden

Segmenteringsmetoden visade att syntetiskt genererad träningsdata kan användas för att träna en modell som identifierar komponenter i en kontrollerad industriell miljö. Den största fördelen med metoden var att stora mängder träningsdata kunde skapas automatiskt, vilket minskade behovet av manuell annotering. Detta gjorde det möjligt att snabbt anpassa modellen till de komponenter som var relevanta för projektet.

Figur 5.4: Exempel på segmentering av verkliga komponenter med modellen som tränades med den syntetiska datapipelinen.

Figur 5.4 visar att modellen kunde segmentera komponenter även i en verklig bild, trots att träningsdatan till stor del genererades syntetiskt. Resultatet indikerar att datapipelinen kan användas för att snabbt ta fram en specialiserad segmenteringsmodell för ett begränsat antal komponenter.

Metoden fungerade bäst när komponenterna var tydligt synliga, hade tillräcklig storlek i bilden och inte överlappade varandra i för hög grad. I dessa fall kunde modellen ge användbara masker som beskrev komponenternas utbredning i bilden. Detta visar att ansatsen är relevant för fortsatt arbete med komponentidentifiering i en kittningsmiljö.

5.5 Begränsningar

Segmenteringsmetoden hade flera begränsningar. Resultatet påverkades av bildkvalitet, kameravinkel, ljusförhållanden och hur väl den syntetiska träningsdatan mot-

svarade verkliga bilder. Små komponenter och detaljer var svårare att segmentera stabilt, särskilt när objektet upptog en liten del av bilden. Stort överlapp mellan komponenter kunde också göra segmenteringen osäker.

En central begränsning var att syntetisk träningsdata inte automatiskt motsvarar verklig data. Även om variationer i skala, rotation och ljus kan införas finns det fortfarande ett sim-to-real gap mellan genererade träningsbilder och verkliga kamerabilder. För att förbättra robustheten behöver metoden därför utvärderas mer systematiskt på verkliga bilder från cellen.

5.6 Vidare möjligheter och alternativa angreppssätt

En fråga som undersöktes var om 3DGS skulle kunna användas för att skapa komponentmodeller och därefter generera träningsdata artificiellt. I praktiken visade det sig dock att upplösningen och detaljkvaliteten i dessa modeller var för låg för de mindre komponenter som ingick i projektet. Viktiga geometriska detaljer gick förlorade, vilket gjorde metoden mindre lämplig för detta ändamål.

Ett alternativ för framtida arbete är att använda CAD-modeller i exempelvis Unity eller Blender för att generera syntetisk träningsdata. En sådan metod skulle kunna ge bättre kontroll över geometri, kameravinklar och scenvariation. Samtidigt ställer den högre krav på realistiska texturer och materialmodeller för att inte öka avståndet mellan simulerad och verklig data. En möjlig vidareutveckling är därför att undersöka om texturer och ytmodeller kan förbättras med bildgenerativa AI-metoder.

5.7 Sammanfattning

Segmenteringsdelen utvecklades för att identifiera och avgränsa komponenter vars position och orientering varierar i lådorna. För detta användes instanssegmentering med en YOLO-baserad modell, eftersom metoden ger pixelvisa masker och kan anpassas till ett begränsat antal komponenttyper.

För att undvika omfattande manuell annotering skapades syntetisk träningsdata genom att filma komponenter mot en magentafärgad bakgrund, extrahera dem med tillhörande masker och placera dem i bilder av tomma lådor med varierad skala, rotation och ljussättning.

Metoden visade att syntetiskt genererad träningsdata kan användas för att träna en specialiserad segmenteringsmodell, men resultatet påverkas av bildkvalitet, objektens storlek och skillnaden mellan syntetiska och verkliga bilder. Fortsatt arbete bör därför fokusera på mer systematisk testning med verkliga bilder och förbättrad träningsdata.

6
Sensorfusion

I detta projekt används sensorfusion för att samla data som upptäckts i videoflöden från de simulerade kameror som implementerats i Unity i ett gemensamt koordinatsystem. Målet i sig är inte att utveckla en avancerad fusionsalgoritm utan att skapa ett enkelt informationsflöde som filtrerar bort dubbletter och där relevanta objekt kan skickas vidare till styrning och visualisering. Detta görs för att minska tapp i prestanda och bibehålla realtidsflödet. Implementationen utvärderas även utifrån dessa delar samt vad som kan utvecklas i framtiden.

6.1 Val av sammansättning av flöde

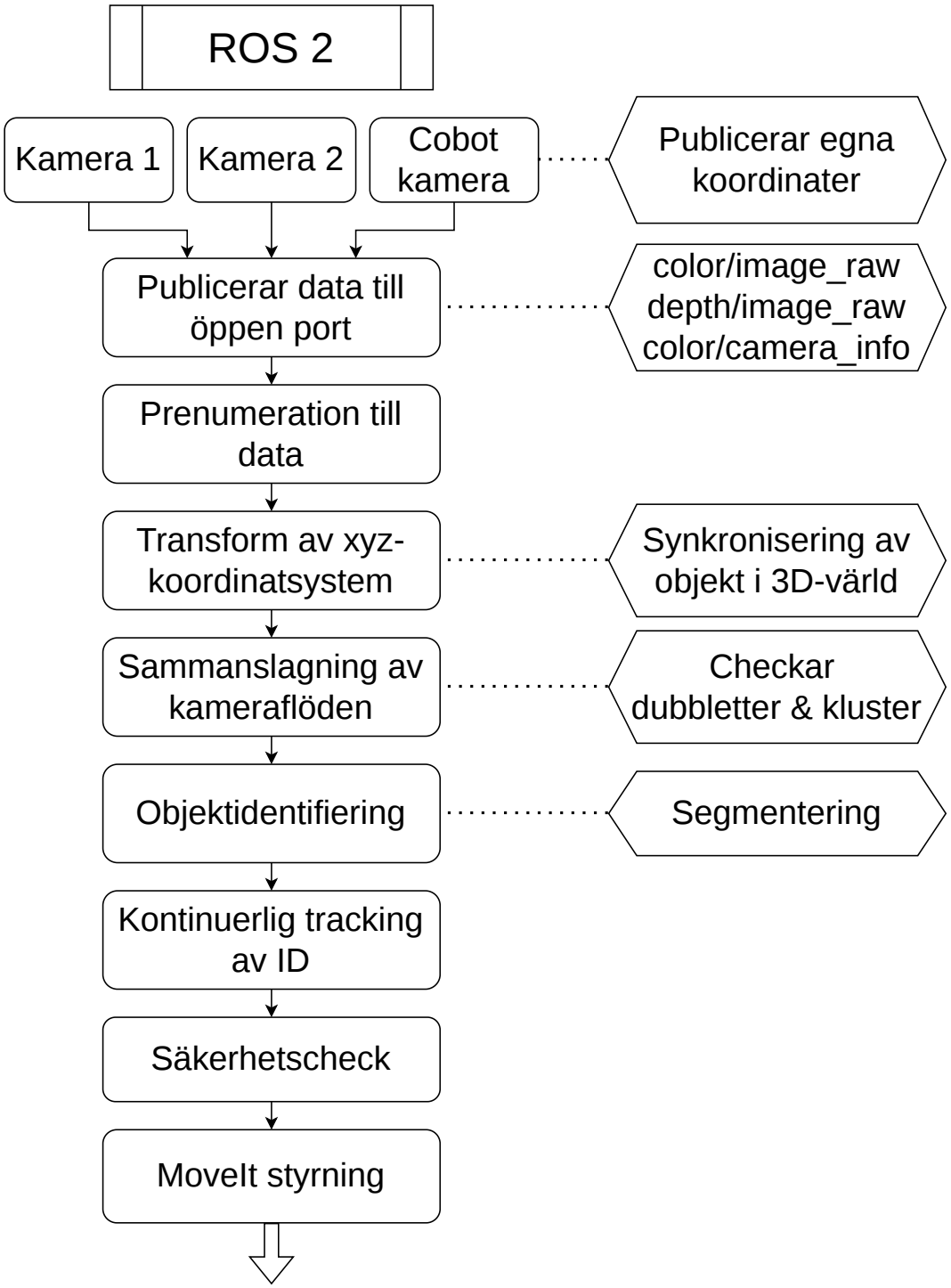
I situationer där det sker flera saker samtidigt krävs det både logik samt säkerhetsprotokoll för att en process ska fungera enligt plan. För att övervaka detta samt kontrollera att allt sker utan olyckor implementeras olika former av sensorer såsom mätinstrument, videoflöde eller andra varianter. Dessa behöver sedan kunna fungera tillsammans utan att duplicera informationen eller skriva över varandra, vilket leder till att funktionen sensorfusion implementeras.

Sensorfusion handlar om att sammanfoga dataflöden från flera sensorer. I detta projekt avser det kamerornas visuella videoflöden [27] [28]. På detta sätt kan en gemensam 3D-värld byggas upp för samtliga sensorer, där ett objekt kan identifieras i samma koordinatsystem oberoende av sensor. Detta gör det möjligt att filtrera bort dubbletter och endast använda den sammanslagna objektinformationen som indata.

När den gemensamma 3D-världen har byggts upp behöver även ny information hanteras i realtid för att kunna användas i systemet. Därför behöver informationsflödet behandlas med låg processtid och samtidigt kunna hantera förändringar när de upptäcks. Detta är viktigt för att bibehålla säkerhetsaspekten när cobot och människa arbetar i ett begränsat område [29] såsom i RITA-cellen.

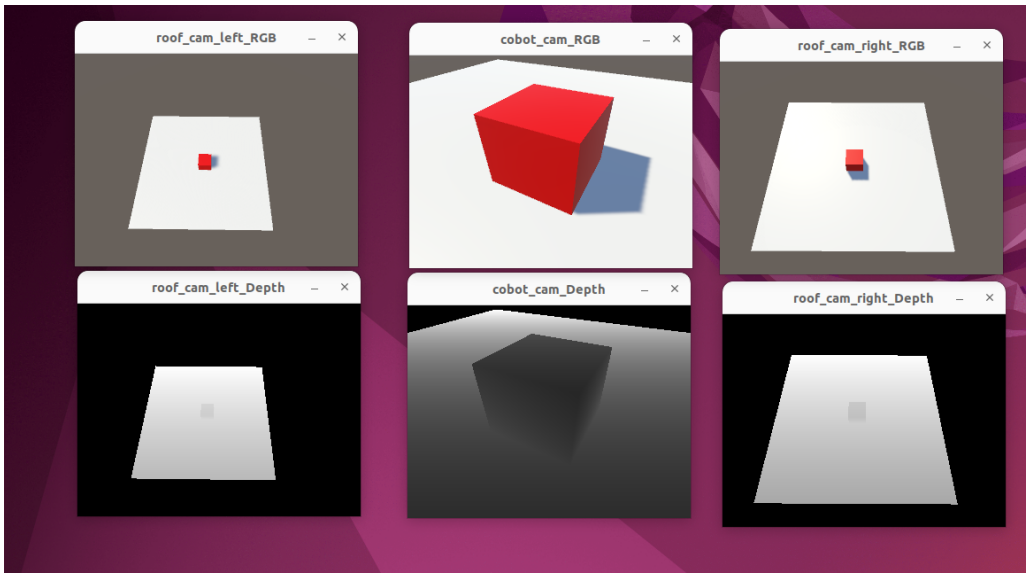
Inom sensorfusion används ofta mer avancerade algoritmer, exempelvis Kalmanfilter eller Bayesiansk inferens [28]. I detta projekt har detta avgränsats till en förenklad metod där medelvärden av sensordata används. Detta kan ge något förvrängd information, vilket har tagits i beaktning eftersom projektets huvudfokus ligger på andra delar av systemet.

6.2 Beskrivning av metod och arbetsflöde



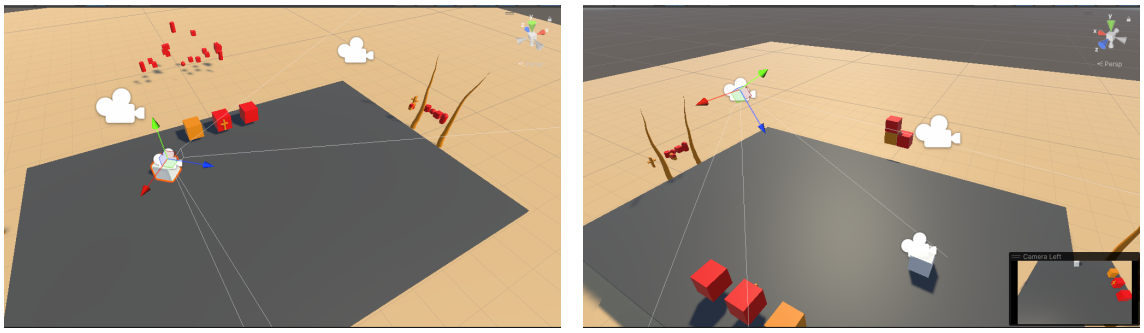
Figur 6.1: Flödesschema över fusionprocessen.

I projektet användes tre videosensorer för att hantera reglering av styrande moment i simuleringen. Kamerorna som användes skickar kontinuerligt dataflöden till systemet, som behöver hantera flera flöden samtidigt. Sensorerna i projektet simulerades i Unity för att efterlikna verkliga kameror med inställningar anpassade efter den miljö som systemet är tänkt att utvecklas för, vilket var två kameror i taket samt en i cobotens huvud. De kameror som simulerades anpassades därför efter en *Red Green Blue* (RGB)-kameras funktion, där djupinformationen utgjordes av syntetisk data baserad på en gråskalad djupkarta i Unity. Den vanliga färginformationen placerades på en skala där ljusare pixlar behandlas som närmare kameran, vilket gör att djupet kan avläsas [30], se figur 6.2. Data från de simulerade kamerorna i Unity skrivs sedan till en öppen port i datorn via paketet ROS-TCP-Connector [21], för att därefter kunna läsas av ROS 2 på samma sätt som presenterades i kapitlet om cobotstyrning.



Figur 6.2: Exempel på hur gråskala läses av i Unity med syntetiska kameror.

För att kunna bygga upp en digital tvilling som stämmer överens både för användaren och för maskinerna behöver denna värld korrigeras för samtliga system, både de visuella systemen som är synliga för människan och maskinernas kamerasensorer. Systemet använder olika koordinatsystem beroende på miljö. I detta fall har Unity koordinaterna $[x, y, z]$, vilket i ROS 2 översätts till $[-y, z, x]$. Genom att transformera data mellan systemen till ett enhetligt koordinatsystem innan sammanslagningen erhålls korrekta koordinater. Detta gör även att speglade kameror kan synkroniseras till samma 3D-världsbild och därmed jämföra upptäckta objekt.



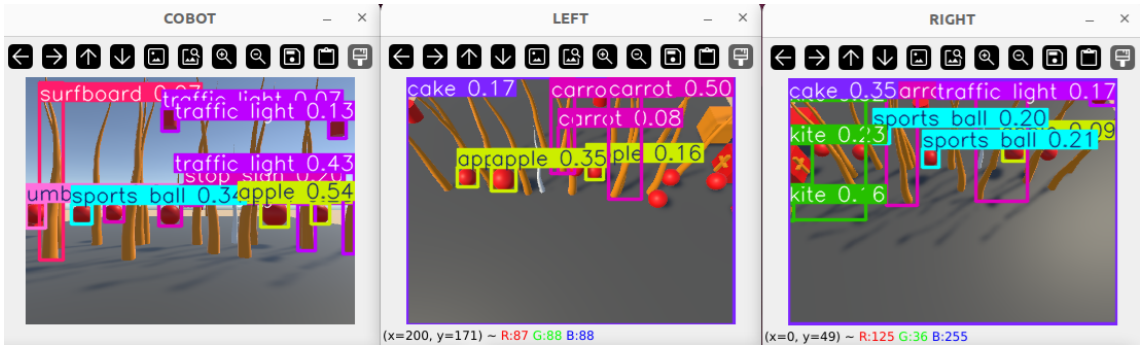
(a) Cobot-kameras vy som uppdateras med dess rörelse i digitala världen.

(b) Tak-kameras vy där det även visas dess ena egna vy.

Figur 6.3: Vyer för kameror som sedda i Unity engine.

När ett gemensamt koordinatsystem har etablerats fortsätter flödet till lokalisering av objekt. Därefter kontrolleras dubbletter, både för att avgöra om kamerornas upptäckta objekt motsvarar samma objekt och om flera närliggande objekt egentligen tillhör samma objekt. Detta görs även genom klustring med *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) [31], där överlappande kluster sammanförs [32]. Objekten kan exempelvis vara objekt med olika former eller människor.

När objekten har upptäckts identifieras de genom YOLO-segmentering enligt tidigare kapitel om segmentering. Vid detektion av objekt krävs det en viss konfidensnivå för klassning av objekt för att inte ta med brus och bakgrund, vilket kan göra att några objekt missas i brusig miljö, se figur 6.4. Varje objekt tilldelas ett globalt ID som bibehålls i systemet. Dessa ID:n spåras sedan kontinuerligt i en lista tills de tas bort. Enligt ett eget val i detta projekt sker borttagning först när ett objekt inte har upptäckts under fem bildrutor [33]. Eventuellt jitter, där ett objekt tillfälligt försvinner under en bildruta, hanteras på detta sätt. Samtidigt tillåts objekt behålla sitt tilldelade ID från segmenteringen utan att ID:t förändras mellan bildrutor eller att systemet upprepade gånger skapar nya objekt.



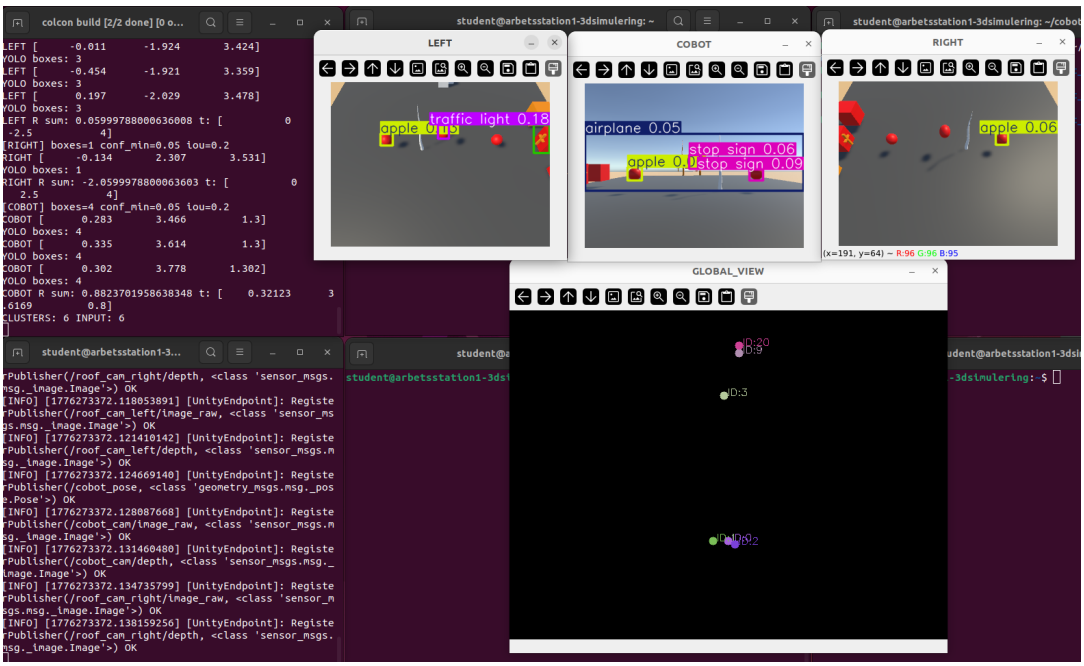
Figur 6.4: Exempel på brusig miljö där många objekt identifieras samtidigt. Detta genom YOLO nanos egna viktparametrar.

Efter identifieringen kan systemet hantera de externa objekt som är relevanta, exem-

pelvis komponenter som ska plockas upp eller objekt som kan innebära en potentiell kollision med systemet. Upplockning av komponenter sker huvudsakligen med hjälp av cobotens egen kamera, eftersom den ger bättre upplösning för detaljer nära komponenten. En enkel brushantering implementerades även för att jämföra avstånd till objekt mellan cobotens kamera och takkamerorna, där cobotens kamera ges högre förtroende vid korta avstånd enligt en förutbestämd gräns baserat på tidigare arbete inom självkörning [34]. En gräns implementerades även för att på korta avstånd ge coboten prioritering i förtroende som är mer detaljorienterad för upptäckta objekt på kortare avstånd.

Under simuleringen publicerar även coboten sina egna koordinater enligt kapitlet om cobotstyrning och kan därmed användas för att identifiera potentiella kollisioner som behöver undvikas. Om ett externt objekt, exempelvis en människa, kommer in i arbetsområdet hanteras detta genom att först skicka en signal till MoveIt [18] om att avbryta. Därefter skapas en zon i området där objektet har upptäckts [35]. Denna tillfälliga zon räknas som fast yta eller kollisionsobjekt, på samma sätt som RITA-cellen. Zonen bibehålls, på samma sätt som globala ID:n, i upp till fem bildrutor efter att objektet inte längre upptäcks på koordinaten. När en zon har etablerats skickas en ny signal till MoveIt för att planera om med den nya kollisionszonen i åtanke.

För att möjliggöra en visuell representation för användaren separeras flödet efter identifieringen återigen till separata flöden innan det visas i ROS 2:s visualiseringsfönster RViz. Där visas kamerans egna videoflöde med YOLO-funktionens ID applicerat.



Figur 6.5: Exempel på kamerornas egna detektioner samt den globala bilden efter att flöden sammansatts och visas i RViz.

6.3 Utvärdering av metoden

Med sensorfusion kan en gemensam världsbild skapas även i den digitala världen, så att sensorerna överensstämmer och ger korrekt funktion för projektets syfte. Detta är relevant både för det ordinarie arbetsflödet och ur säkerhetssynpunkt, eftersom systemet ska kunna förebygga olyckor som förutsatts till en av de viktigare delarna. Ett enhetligt koordinatsystem för samtliga funktioner är en viktig del för att skapa ett enhetligt arbetsflöde. Detta gäller särskilt säkerhetsaspekten, eftersom människors position ständigt kan förändras och förebyggande funktioner därför är prioriterade.

Möjligheten att använda flera sensorer ger dessutom mer information för att upptäcka problem. Sammansättningen gör att potentiella dubbletter kan upptäckas och filtreras innan resultatet skickas vidare i processen. Den ger även en mer korrekt bild av arbetsområdet, eftersom flera vinklar kan användas för att uppskatta objektens position bättre än vad en enda vinkel kan. Detta kan även stödja coboten i dess planering, även om behovet är mindre vid själva upplockningen av objekt. Om något täcker en sensor finns fortfarande information från övriga sensorer som kan fylla i delar av det synfält som missas, vilket gör att faror som annars hade missats kan upptäckas. Om en sensor fallerar kan detta dessutom kompenseras genom överlapp från övriga sensorer som tillsammans övervakar området. Därmed kan arbetsstationen säkras utan ett överflödigt informationsflöde som riskerar att få automatiska funktioner att reagera felaktigt.

Däremot bör det beaktas att avsaknaden av Kalmanfilter gör att data i sig inte helt stämmer överens med övriga koordinatsystemet, vilket kan leda till problem som att marginalerna i uträkning kan leda till mindre kollision ändå eller att cobotens egna koordinater kan vara felaktiga mot verkligheten. Delsystemet har endast lyckats testas i ett eget isolerat fall och bör därmed tas i beaktning att det kan skapa problem då det integreras i resterande delar.

Något som även märktes var att om det blev många objekt samtidigt som hade ID:n så märktes flaskhalsen i flödet då brusiga miljöer introducerades som i figur 6.4. Där mättes det att upp till tre sekunders fördröjning vid rörelse av cobot-kamera jämfört med vanligt, men det kan även ha berott på andra omständigheter som inte kunde uteslutas. Svårigheter fanns i att få tak-kamerorna och cobot-kameran att ha ett gemensamt koordinatsystem då två är statiska och en är rörlig i systemet som roterar, vilket ökar komplexiteten.

6.4 Framtida utveckling av metod

Sensorfusion har i detta projekt endast använts på en grundläggande nivå. Eftersom fusionen avgränsades till användning av medelvärden kan datan bli något förvrängd. Detta hade kunnat lösas bättre genom implementering av exempelvis Kalmanfilter

eller andra mer avancerade algoritmer. Sådana metoder är vanligt förekommande i liknande processer och rekommenderas därför för framtida utveckling.

Eftersom projektet baseras på en simulering av RITA-cellen, med syfte att testa metoden innan den tillämpas i verkligheten, finns möjlighet att i framtiden använda RGB-D-kameror direkt. Detta skulle ge mer exakta djupvärden för objekt. Genom att använda kamerornas egna mätningar av avstånd, i stället för gråskalevärden som uppskattar avstånd baserat på nyanser, kan mer korrekta värden erhållas och användas i algoritmerna.

Ur ett säkerhetsperspektiv hade ett kollisionsprotokoll med fokus på förutsägelse av rörelser även kunnat implementeras. Om ett objekt exempelvis rör sig snabbt in i arbetsområdet skulle systemet vid första upptäckt kunna skapa en förutspådd bana och hantera situationen innan risk för kollision uppstår [36]. Detta är främst relevant för marginalberäkning och effektivitet, eftersom det skulle kunna låta coboten arbeta i mindre utrymmen samt fortsätta arbeta i mer aktiva miljöer utan att stanna upp varje gång en människa befinner sig i samma område.

6.5 Sammanfattning

Genom sensorfusion kan samtliga sensorer tillföra sin information som hanterar dubbletter och endast ger nödvändig information till resterande system där denna sammanflätats. Detta är något väsentligt för att kunna användas i ett gemensamt koordinatsystem och på så sätt bygga upp en enhetlig digital värld.

I detta projekt är det väldigt få sensorer som ska sättas ihop men om utvecklingen sker exponentiellt så är det ett måste att få det korrekt, redan på låg nivå, vilket oftast sker i stora fabriker. Att dessutom kunna utnyttja flera sensorer med överlapp ökar effekten från samtliga då mer information kan utvinnas så länge den hanteras på ett lämpligt sätt.

Det filtrerar även till en viss nivå vad som är nya objekt och hur länge dessa ska spåras, vilket kan minimera beräkningskraften på hårdvara som blir en flaskhals vid stora mängder data. I detta projekt hölls även sensorfusionen till en enkel nivå och hade även behövt de mer avancerade filtren som finns ute idag för att öka igenkänning.

När dessa objekt väl tas med så möjliggör det för systemet att upptäcka objekt och tilldela globala ID:n som sedan kan användas för säkerhetsprotokoll och andra funktioner. Det är viktigt för att förebygga olyckor men även för att utföra det vanliga arbetet som att tilldela korrekta ID:n och då kunna utföra upplockning av komponent. Sensorerna är även de som hanterar säkerheten på arbetsstationen och behöver därmed ha hög prioritet för att kunna avbryta styrningen. Därmed krävs det att dessa har en korrekt implementering och att sammansättningen av flödena gjorts på rätt sätt.

7
Interaktion med simuleringsmiljön

Interaktion med den digitala tvillingen är en central komponent eftersom det möjliggör analys, styrning och förståelse av systemet i realtid. I detta projekt föll valet av interaktion till VR, eftersom detta gör att användaren kan uppleva och manipulera miljön på ett mer intuitivt och immersivt sätt än genom andra mer traditionella interaktionsmetoder. I detta kapitel behandlas därmed valet av VR, hur implementationen går till och huruvida VR interaktionen fungerade i simuleringsmiljön.

7.1 Val av VR som interaktionsmetod

För att möjliggöra en mer intuitiv och rumsligt korrekt förståelse av den simulerade miljön föll valet på VR som interaktionsmetod. I jämförelse med det mer traditionella alternativet att styra och interagera genom skärm och mus, kan man i VR uppleva och interagera med miljön i tre dimensioner. Detta medför att användaren får röra sig och manipulera systemet på ett mer verkligt och naturligt sätt.

I detta projekt är det särskilt viktigt att få en realistisk bild av systemet då målet är att sänka sim-to-real gapet och att människan ska kunna samarbeta med coboten på en gemensam yta på ett säkert sätt. Även om detta går att visualisera och testa genom annan mer traditionell interaktionsmetod blir VR mer realistisk för användaren och det är lättare att uppfatta relationer mellan objekt, avstånd och andra dynamiska förändringar i systemet.

I VR är det även lättare att interagera på ett naturligt sätt, såsom att greppa, peka och manipulera direkt i miljön. Detta gör att VR kan användas för att testa olika scenarier i realtid, till exempel flytta objekt, ändra inställningar eller simulera problem. Alltså kan VR användas som ett testningsverktyg och inte bara för att visualisera miljöer. Med detta fås även en grund för beslutsfattande, då det är lättare att upptäcka fel, ineffektivitet eller risker i systemet när miljön och upplevelsen blir mer realistisk. Alltså kan VR bidra till att rätt beslut tas innan något byggs eller ändras i verkligheten.

För framtida bruk när ett system är effektiviserat och implementerat i verkligheten kan VR även användas i ett utbildningssyfte. Istället för att träna personal i den verkliga miljön kan man börja med att introducera simuleringsmiljön, där de får uppleva och arbeta i systemet utan risk. Inlärningsprocessen skulle därmed bli mer effektiv i och med att antal risker och fel minskar då personal blivit van vid systemet innan de testar det i verkligheten. I ett utvecklings- och samarbetsperspektiv kan VR även i framtiden användas för att mötas i den digitala tvillingen. Vid brådskande ärenden som gör att alla parter inte kan samlas på samma plats i tid kan man då genom VR gå in i simuleringsmiljön och diskutera problem och eventuella lösningar.

Självklart finns det även brister med VR. Till att börja med kan den digitala tvillingen vara väldigt tung med mycket data, objekt och simuleringar, som i sin tur påverkar prestandan på VR. För att få en bra upplösning i VR krävs det en stabil bildfrekvens och därför kan den digitala tvillingen behöva förenklas något för att uppnå det, i sin tur påverkar ju detta detaljnivån på miljön och simuleringen. Vidare kan användarinteraktionens precision vara begränsad i jämförelse med andra interaktionsmetoder, vilket kan skapa problem i detaljerad manipulation i den digitala tvillingen. I andra metoder går det även att visa flera informationskanaler samtidigt, vilket blir en utmaning i VR eftersom det endast går att visa ett kameraflöde åt gången som då leder till att man inte kan presentera lika stora mängder information på ett överskådligt sätt. Slutligen kan användarkomforten påverkas vid längre användning av VR. Detta kan utspela sig som fysisk utmattning eller illamående, som då begränsar systemets användning över tid.

Dessa begränsningar har beaktats i projektet och vägs mot de fördelar VR erbjuder i form av ökad förståelse och interaktivitet. Fördelarna anses väga tyngre i relation till projektets övergripande mål, där VR möjliggör en mer naturlig interaktion samt en djupare förståelse av den digitala tvillingens struktur och beteende än vad andra interaktionsmetoder erbjuder.

7.2 Implementering av VR i Unity

Implementeringen av VR genomfördes i grafikmotorn Unity med hjälp av dess inbyggda stöd för XR-utveckling. För att möjliggöra kommunikation mellan Unity och VR-hårdvaran användes runtime-miljön SteamVR tillsammans med Vive Hub, vilket hanterar anslutning, spårning och konfigurering av headset och kontroller. För att skapa en mer verklighetstrogen och immersiv interaktion integrerades VR-systemet med en humanoid avatar som representerar människan i samarbetet med coboten.

7.2.1 Unity paket, SteamVR och Vive Hub

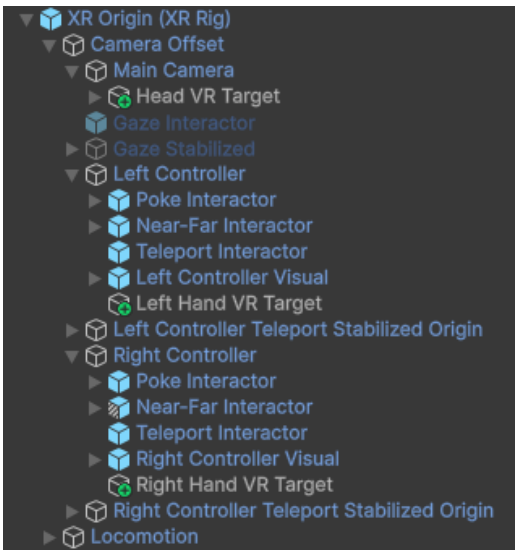
Implementeringen av VR i Unity bygger på flera paket och system inom Unitys XR-ekosystem. För att möjliggöra VR-funktionalitet krävs först att XR-stöd aktiveras och konfigureras i projektet. Implementationen baserades delvis på tidigare demonstrationsmaterial [37] och dokumentation för Unitys XR-system [38].

Den mest centrala komponenten för VR-implementeringen är XR Plugin Management, eftersom det är genom denna som man kan installera och aktivera leverantörplugin för den enhet man använder [38]. I detta projekt användes OpenXR som leverantörs-plugin, då detta stödjer Vive-headset samt möjliggör en mer hårdvaruoberoende implementation [39]. För att säkerställa korrekt kommunikation mellan systemet och VR-kontrollerna användes HTC Vive Controller Profile inom OpenXR-konfigurationen.

För interaktion och rörelse i den virtuella miljön användes XR Interaction Toolkit. Detta paket tillhandahåller färdiga komponenter för funktioner såsom objektinteraktion, greppning och anpassningsbart rörelsesystem [38]. Utöver detta användes tillhörande Starter Assets för att förenkla implementationen av grundläggande interaktionsfunktioner och locomotion-system.

Kommunikationen mellan Unity och VR-hårdvaran hanterades genom runtime-miljön SteamVR. SteamVR ansvarar för spårning av headset och kontroller för att få rätt position och rotation i simuleringsmiljön, samt rendering av bilden och hantering av användarinput [40]. För konfigurering och hantering av Vive-headsetet användes även Vive Hub, vilket möjliggör anslutning, drivrutinshantering och kalibrering av systemet [41][42]. Tillsammans säkerställer dessa system att VR-utrustningen fungerar korrekt tillsammans med Unity-applikationen.

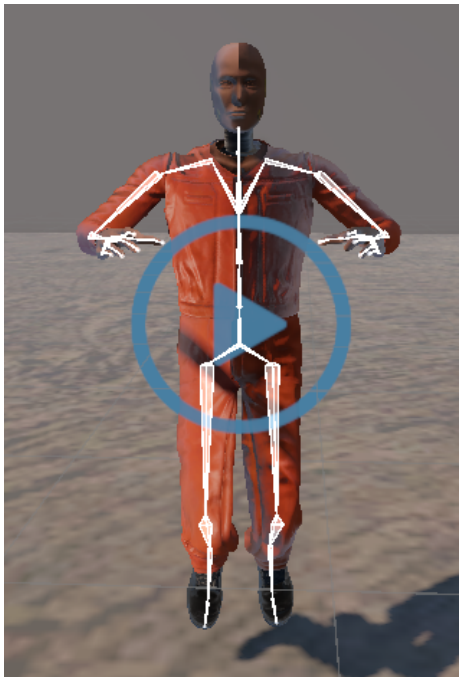
Som grund för användarrepresentationen i VR användes en XR Origin. XR Origin representerar användaren i den virtuella miljön och hanterar huvudkamera, kontroller, interaktionskomponenter och förflyttning [37], se figur 7.1 som visar XR Origins uppbyggnad. Denna struktur utgör grunden för kopplingen mellan användarens fysiska rörelser och motsvarande rörelser i den virtuella miljön.



Figur 7.1: Bild i Unity som visar XR Origin trädet och dess viktigaste funktioner såsom huvudkameran och kontrollerna, samt VR targets som kopplar samman delarna mot avataren.

7.2.2 Sammankoppling av VR och avatar

För att minska utvecklingstid och svårigheten att implementera en avatar i Unity användes en färdig humanoid avatar med fördefinierat skelett och animationer [43], se figur 7.2. Avataren importerades till Unity i FBX-format och ställdes in för att användas med Unitys humanoida riggsystem. Implementationen av avatarens rörelser och kopplingen till VR-systemet baserades på en video som visar processflödet [44].



(a) Figuren visar avataren i VR-miljön och det inbyggda skelettet

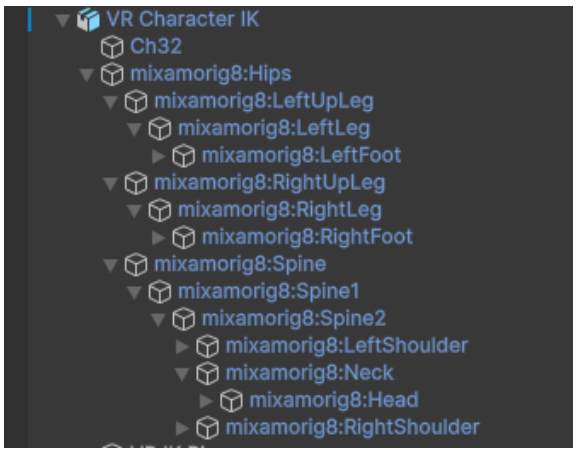
(b) Figuren visar den humanoida avataren innan implementation i sin grundposition.

Figur 7.2: Två bilder av avataren som användes i projektet.

För att möjliggöra avatarens kroppsrörelser användes *inverse kinematics* (IK), vilket innebär att avatarens leder och kroppsdelar följsamt anpassas efter VR headsetet och kontrollerna genom att ta dess positioner och med matematiska beräkningar få ut rotationer [45]. Till detta användes Unitys paketet Animation Rigging, vilket möjliggör konstruktion av riggar och constraints för styrning av avatarens rörelser [46].

En separat IK-rigg skapades för avataren där armar, ben och huvud kunde styras individuellt. För armarna implementerades Two Bone IK Constraints, där händerna kopplades till target-objekt som senare följde användarens VR-kontroller. Detta gjorde att armrörelserna utförda med kontrollerna i verkligheten kunde synkroniseras med avataren i simuleringsmiljön. Hint-objekten positionerades för att styra armarnas böjning och säkerställa mer naturliga rörelser. Se figur 7.3 för strukturen av IK-riggen.

(a) VR IK Rig för armar och ben med deras tillhörande target och hint objekt.



(b) VR Character IK dit respektive ben och arm target kopplas till rätt kroppsdel i skelettet.

Figur 7.3: Illustration av VR IK Rig och VR Character IK som samarbetar med varandra för att avatarens rörelser ska stämma överens med kontrollerna.

Även benens rörelser implementerades med hjälp av IK, se figur 7.3. För att förbättra fötternas placering och kontakt med marken användes ett externt Foot Solver-script [47]. Detta möjliggjorde att fötternas position och rotation kunde anpassas i förhållande till underlaget, vilket bidrog till mer realistiska rörelser och minskade risken för att avatarens fötter gick igenom golvet. Manuell justering av offset-värden och target-positioner krävdes i systemet för att uppnå stabila och naturliga rörelser.

För huvudets rörelser användes en constraint-baserad lösning där avatarens huvud ständigt följde positionen och rotationen från VR-headsetet. På motsvarande sätt kopplades avatarens händer till separata VR-targets kopplade med respektive handkontroll, se figur 7.1 för strukturen av VR-targets i XR Origin. Genom denna struktur kunde hela avatarens överkropp röra sig i enlighet med användarens verkliga rörelser i VR-systemet.

För att kunna interagera med objekt implementerades handanimationer och greppfunktioner kopplade till kontrollerna. Genom att läsa av input-signaler från VR-kontrollerna kunde avatarens händer växla mellan pinch- och grab-animationer. Detta möjliggjorde mer naturliga interaktioner med objekt i simuleringsmiljön, vilket var en passande funktion med avseende på kittnings-aspekten i projektet.

7.2.3 Tekniska utmaningar

En av de största tekniska utmaningarna var implementationen av IK tillsammans med VR-spårning. Eftersom avatarens rörelser styrdes av headset och kontroller uppstod problem med naturliga kroppsrörelser, särskilt vid arm- och benpositionering.

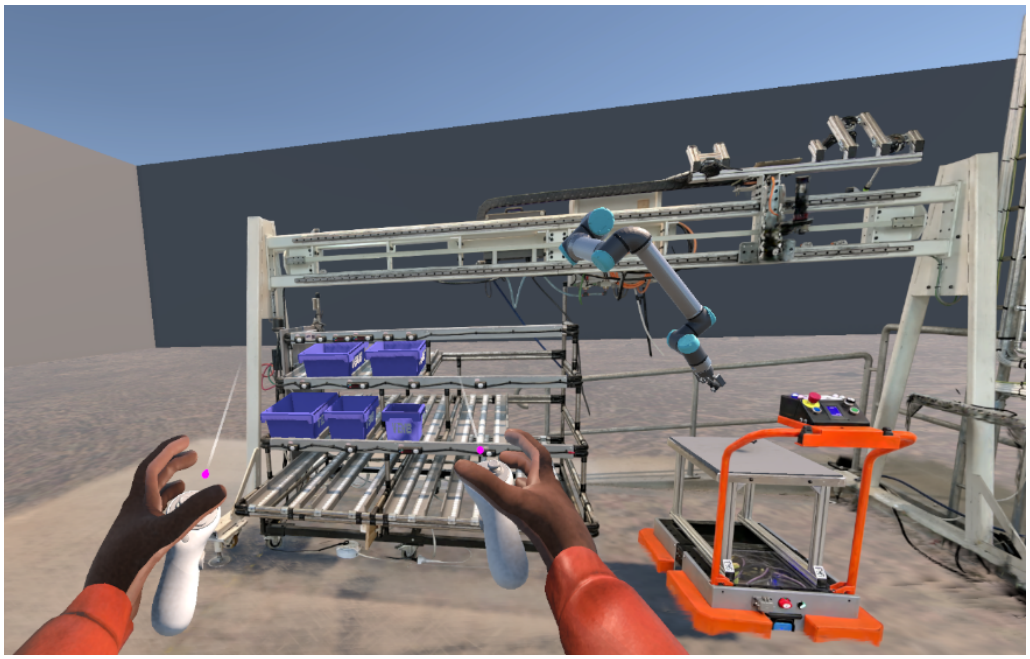
Ytterligare utmaningar uppstod vid synkronisering mellan avatarens huvud och huvudkameran i XR Origin. Fel positionering och offset-värden ledde till att huvudet inte följde användarens rörelser korrekt, vilket påverkade realism och användarupplevelse. För att minska dessa problem krävdes kalibrering av target-positioner och kroppsoffset i avatarens rigg.

En annan teknisk utmaning var renderingen av bilden i VR-headsetet. Under implementationen uppstod problem där rendering endast fungerade korrekt i ett av headsetets ögon. Problemet grundade sig i renderingsinställningarna för XR och löstes genom att ändra renderingsläget till Multi-Pass rendering, vilket möjliggjorde separat rendering för respektive öga och resulterade i korrekt visualisering i headsetet.

7.3 Utvärdering av VR i simuleringsmiljön

Resultatet av VR-implementeringen i den digitala tvillingen var övergripande positivt, men varierade beroende på vilken del av systemet som analyserades. Själva VR-miljön fungerade stabilt och möjliggjorde en tydlig visualisering av den splattade RITA-cellen med hög detaljnivå och god rumslig uppfattning. Detta bidrog till en mer immersiv användarupplevelse jämfört med traditionella interaktionsmetoder och gjorde det lättare att analysera relationer mellan objekt, avstånd och rörelser i systemet.

Den virtuella miljön gav även en bättre förståelse för hur användaren och coboten delar arbetsytan, vilket är centralt ur både samarbets- och säkerhetsperspektiv. Genom att kunna röra sig naturligt i miljön och observera systemet ur ett mänskligt perspektiv kunde VR-lösningen bidra till att minska sim-to-real-gapet, åtminstone ur ett visuellt och rumsligt perspektiv. Se figur 7.4 för användarens perspektiv av simuleringsmiljön.

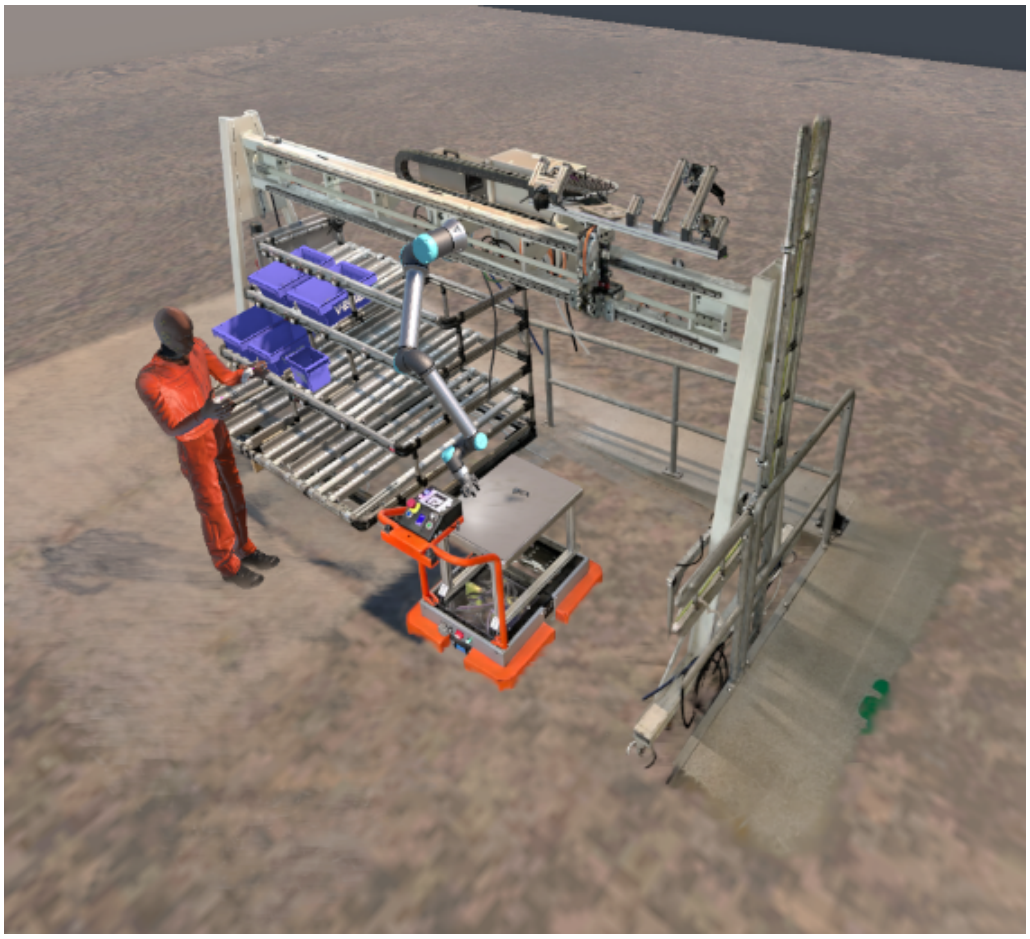


Figur 7.4: Användarens perspektiv av simuleringsmiljön.

Samtidigt identifierades flera begränsningar i implementationen av avataren. Avataren fungerade som en bra representation av användaren, men rörelserna upplevdes inte helt naturliga eller verklighetstrogna. Detta berodde bland annat på begränsningar i IK-systemet och den relativt enkla kopplingen mellan VR-spårning och avatarens kroppsrörelser. Trots detta kunde avataren genomföra sitt generella syfte att visualisera samarbetet mellan människa och cobot på ett bra sätt i den digitala tvillingen.

Även greppfunktionen fungerade i grundläggande form, där användaren kunde manipulera objekt i miljön genom VR-kontrollerna. Funktionaliteten testades dock endast ytligt och inte i relation till kittningsarbetet som ska utföras i cellen. Det är därför svårt att veta hur väl interaktionen hade fungerat vid mer komplexa arbetsmoment, såsom att plocka komponenter eller flytta lådor i miljön. Om systemet ska kunna användas i verkliga produktionssammanhang krävs sannolikt vidare utveckling och mer användartester.

Trots dessa begränsningar visar resultatet att VR är en lovande metod för interaktion med digitala tvillingar, särskilt inom människa–robot-samarbete. Kombinationen av immersiv visualisering och möjlighet till naturlig interaktion skapar goda förutsättningar för framtida utveckling inom simulering, säkerhetsanalys och utbildning. Se figur 7.5 för helhetsbild av avataren i simuleringsmiljön.



Figur 7.5: Helhetsbild av avataren i RITA-cellen.

7.4 Sammanfattning

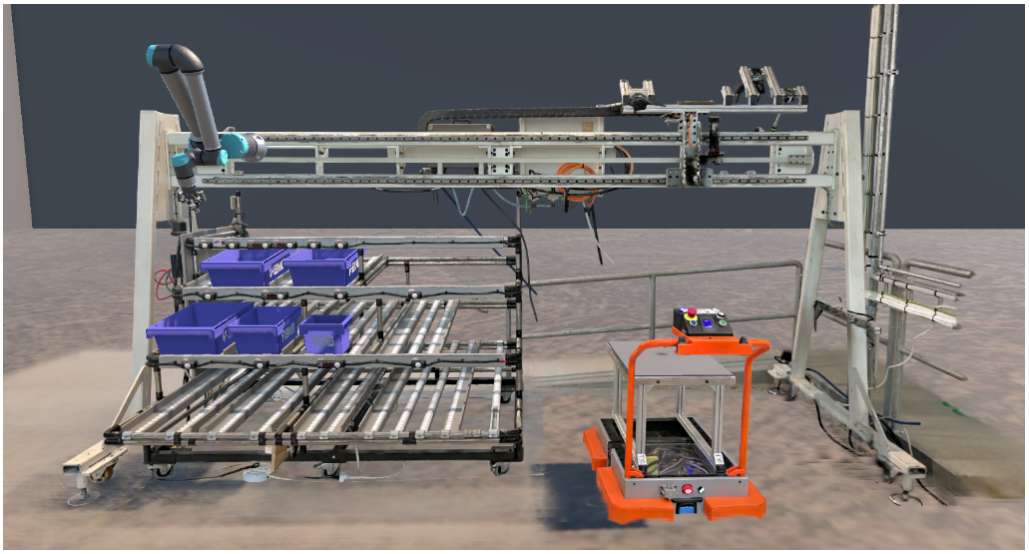
VR användes som interaktionsmetod i projektet för att skapa en mer immersiv och naturlig interaktion med den digitala. Med Unitys inbyggda XR-system, SteamVR och Vive Hub implementerades stöd för VR-headset och kontroller, vilket möjliggjorde interaktion och navigering av simuleringsmiljön. För mänskligrepresentation i den digitala tvillingen implementerades även en humanoid avatar som genom IK och VR-spårning kunde styras.

Resultatet visade att VR fungerade bra för visualisering och analys av den digitala tvillingen. Med en tredimensionella miljön får användaren en bättre rumslig förståelse och analys av relationen mellan objekt, människor och coboten. Samtidigt identifierades vissa begränsningar främst kopplade till avatarens rörelser och interaktionsfunktioner. Trots dessa utmaningar bedöms VR vara en lovande metod för framtida arbete inom digitala tvillingar och människa–robot-samarbete.

8

Validering

Detta kapitel analyserar och validerar resultaten av arbetet, se figur 8.1 för resultat av den digitala tvillingen. Utvärderingen genomförs utifrån fotorealismen, funktionalitet och av sim-to-real gapet mellan simulering och verklighet. Vidare diskuteras begränsningar, felkällor, säkerhetsavvägningar och etiska aspekter.



Figur 8.1: Resultat av den digitala tvillingen i Unity med alla implementerade delar. 3DGS-modellen, styrbar cobot, blå CAD-lådor med komponenter och AMR där komponenter placeras.

8.1 Tolkning av resultat

Arbetet har resulterat i en fungerande grund för hur 3DGS kan användas vid framtagning av en digital tvilling för en industriell robotcell. Resultatet bör dock inte tolkas som en färdig produkt, utan snarare som en prototyp där flera centrala delsystem har utvecklats och testats. I detta projekt är inte alla delproblem fullt implementerade i den slutliga simuleringsmiljön. Styrningen och sensorfusionen implementerades initialt på en Linux-dator, medan den slutliga simuleringsmiljön kördes på en Windows-dator. Detta skapade integrationsproblem som gjorde att dessa delar inte kunde kopplas in fullt ut i slutdemonstrationen. I stället skapades en visuell demo med hårdkodad rörelse för att visa den avsedda funktionen och målet med arbetet. Denna demovideo kan ses via länken i appendix A.1 Google Drive länk.

Vid tolkning av de enskilda delresultaten framgår flera tydliga framsteg. Först och främst har arbetet lyckats ta fram en fotorealistisk miljö med hjälp av 3DGS. Denna miljö utgör en central grund för resten av arbetet, eftersom en realistisk simuleringsmiljö är en förutsättning för att kunna minska sim-to-real gapet. Samtidigt är miljön inte felfri. En viktig begränsning är att 3DGS i nuläget inte skapar en direkt användbar mesh, vilket innebär att modellen inte kan användas direkt för kollisionsdetektion eller fysiksimulering. Detta skulle kunna hanteras genom att skapa en separat mesh av RITA-cellen, exempelvis i CAD, och använda denna som en dold kollisionsmodell i simuleringsmiljön. På så sätt kan den fotorealistiska 3DGS-modellen användas för visualisering, medan den dolda meshmodellen används för kollisioner.

Resultatet visar även att Unity kan användas som en integrationsplattform för att skapa en simuleringsmiljö som i viss utsträckning kan kontrolleras på ett liknande sätt som den verkliga miljön. Genom ROS 2 skapas en kommunikationsstruktur som teoretiskt gör det möjligt att använda liknande styrflöden för både den simulerade och den verkliga robotmiljön. Arbetet visar att ett fungerande flöde mellan Unity och ROS 2 kan etableras, där målpositioner kan skickas från Unity till ROS 2, planering kan utföras i MoveIt och den resulterande trajektorian därefter kan utföras i simuleringsmiljön.

En fungerande, men ännu inte optimerad, banplaneringslösning implementerades också under arbetet. Denna lösning gjorde det möjligt att testa kommunikationsflödet mellan ROS 2, MoveIt och Unity. Resultatet visar att roboten kan styras i simuleringen utifrån planerade trajektorier, vilket är ett viktigt steg mot en mer realistisk digital tvilling. Däremot finns fortfarande begränsningar. Rörelserna är inte alltid optimerade, och kollisionshanteringen är inte fullständig. Eftersom 3DGS-modellen inte ger en användbar kollisionsmesh kan kollisioner med RITA-cellen inte beräknas direkt utifrån den fotorealistiska modellen. Dessutom visade arbetet att beräkningstiden ökade och planeringen blev mindre pålitlig när mer detaljerade kollisionsobjekt inkluderades i MoveIt.

Resultatet från segmenteringen är också positivt. Det tyder på att syntetiskt genererad träningsdata kan användas för att träna en segmenteringsmodell som identifierar komponenter i en kontrollerad miljö. Detta är relevant eftersom det minskar behovet av omfattande manuell annotering av träningsdata. Samtidigt finns det fortfarande begränsningar. Verklig träningsdata har fortfarande fördelar, eftersom den bättre representerar variationer i ljus, kamerakvalitet, bakgrund och objektens faktiska utseende. Det är dessutom fortfarande oklart hur väl segmenteringsmodellen fungerar i realtid i en verklig industriell miljö, där förhållandena kan vara mer varierande än i den simulerade eller syntetiskt genererade datan.

Implementeringen av sensorfusion möjliggjorde att flera sensorer kunde kopplas in till samma system och även skapade av ett övergripande säkerhetsnät vad gäller arbetsområdet. Detta är dock ett av de områden med potential för existerande ut-

veckling där användbarheten kan öka kraftigt men då att komplexiteten följer med. Att bibehålla den informationen som fås via sensorer är något som kan ha stor inverkan i systemen som bygger på det. Därmed fanns det stor användning av denna process i projektet men kunde göras bättre med avancerade filter för att uppnå den tillämpningsnivå som eftersträvades.

Resultaten från VR-implementeringen tyder på att en immersiv interaktion, såsom VR, kan bidra till att en förbättrad rumslig förståelse av den digitala tvillingen. Eftersom användaren kunde röra sig i miljön och därmed observera systemet ur ett mänskligt perspektiv kunde analyser av objekt, avstånd och cobotens rörelser lättare genomföras. I och med att projektet arbetar mot en säkrare robot-människa-interaktion är denna aspekt central för att förstå gemensamma arbetsytor och säkerhetsavstånd. Resultaten indikerar även att VR kan bidra till att minska sim-to-real-gapet ur ett visuellt och rumsligt perspektiv, samtidigt är flera aspekter kopplade till fysisk realism fortfarande begränsade. Problem som identifierades grundar sig främst i avatarens rörelsemönster som är onaturligt, vilket tyder på fortsatta utmaningar i realistisk mänsklig representation när det kommer till inverse kinematics. I framtiden hade detta haft stor påverkan på realism och användbarhet vid mer avancerade arbetsmoment.

8.2 Validering mot frågeställningarna

Resultaten visar att 3DGS är en välfungerande metod vid skapandet av visuellt realistiska digitala tvillingar av industrimiljöer. Den splattade modellen kunde genom snabb rekonstruktion integreras i Unity och därefter användas tillsammans med cobotstyrning, perception och VR. Simuleringsmiljön blev visuellt realistisk vilket möjliggjorde analys av relationer mellan objekt, coboten och människan.

Vissa begränsningar identifierades dock med metoden. 3DGS-modellen saknade mesh-geometri vilket gjorde att den inte kunde användas direkt som kollisionsmodell i MoveIt, som i sin följd begränsade användningen vid banplaneringen och fysikbaserade simuleringar. Detta tyder på att 3DGS lämpar sig bäst för visuell realism snarare än fysisk simulering.

Det utförda arbetet visar också att den digitala tvillingen kan användas som en plattform för simulering och utveckling av robotapplikationer. Integrationen mellan Unity och ROS 2 skapade ett fungerande system där målpositioner som genererades i Unity kunde utföras av coboten genom beräknade trajektoria. Trots huvudsakligen syntetisk genererad träningsdata visar resultaten även att segmenteringsmodellen kunde identifiera komponenter i verkliga bilder. Implementationen av VR bidrog även till att användaren får en bättre rumslig förståelse av arbetsmiljön samt gjorde det möjligt att analysera människa-robot-interaktion på ett mer intuitivt sätt. Resultaten tyder därmed på att digitala tvillingar kan vara användbara för simulering och utveckling av kollaborativa robotsystem.

Även om resultaten är lovande finns vissa begränsningar. Systemet är inte tillräckligt

utvecklat för industriell användning, då flera delar bygger på förenklade lösningar. Till exempel användes inte fysikbaserad gripning och somliga kollisionsobjekt kunde inte hanteras korrekt i MoveIt. I och med detta bör resultaten ses som ett proof of concept snarare än en färdig lösning.

8.3 Begränsningar

Under projektet har flera begränsningar identifierats som påverkar resultatet och valideringen. En teknisk begränsning är att den digitala tvillingen främst utvärderas utifrån visuell realism eftersom 3DGS inte genererar meshgeometri för fysikbaserad simulering. Detta gör att fysisk simulering av gripande och banplanering inte inkluderades i projektets avgränsningar. Begränsningar i hårdvaran har påverkat tidsåtgången för renderingen av 3DGS, bildfrekvensen och kvaliteten vid filmning samt begränsat minnesresurser vid träning av segmenteringsmodeller.

På grund av sekretess från Volvo vid filmning kunde inte RITA-cellen placeras i en verklig industriell miljö, därav har cellen placerats i en syntetisk miljö. Vidare fick inga referensbilder samlas in vilket medför begränsningar för utvärderingen. Gruppens begränsade erfarenhet av Unity tillsammans med projektets tidsram har påverkat utvecklingsprocessen som orsakade ett begränsat antal funktioner och endast ett scenario för miljön. Som resultat av tidsramen kunde inte samtliga delproblem implementeras vilket begränsar möjlighet för validering av systemet som helhet.

Den begränsade mängden träningsdata från variationer av miljöer innebär att modellens robusthet inte kan testas för ljusvariationer, objektplaceringar och andra scener.

Som slutsats har flera begränsningar påverkat resultaten vilket medför resultaten bör tolkas med viss försiktighet. Trots detta indikerar resultaten god potential för vidareutveckling av systemet.

8.4 Felkällor

En felkälla i projektet var hanteringen av filformat vid export av 3DGS-modellen. Den 3DGS-modell i resultatet av projektet bygger på en egenformaterad version vilket nämns i kapitel 3.2.2, vilket innebär att information kan ha förändrats eller gått förlorad i konverteringssteget. Exempelvis kan detaljer i färg, position, orientering, opacitet eller andra splatt-parametrar ha påverkats. Inga större visuella skillnader upptäcktes men detaljer som missats kan ha gått förlorade.

Resultatet påverkades även av inspelningsstrategin. Exempelvis ifall gånghastigheten var för hög, kamerans vinkel förändrades eller att en yta inte fångats från tillräckligt många vinklar. Golvytor var särskilt svåra att fånga, eftersom kameran hade begränsningar kring att vinklas och har i den slutliga modellen lägre kvalitet på nära håll än exempelvis balkarna på cellen. Ljusförhållanden kan också ha påver-

kat resultatet. Även om belysningen på RITA-cellen ansågs jämn förekom mörkare ljussättning av balkar längre ner i strukturen vilket kan ha bidragit till sämre rekonstruktion. Rörliga objekt i scenen kan dessutom ha orsakat artefakter.

En ytterligare felkälla är segmenteringen eftersom den endast tränades på syntetiskt genererad träningsdata. Skillnaden mellan syntetiska och verkliga bilder kan ha påverkat resultatet och faktorer som ljussättning, kameravinkel, bildkvalitet, skuggor och överlapp mellan komponenter kan påverka hur väl modellen segmenterar objekt.

Sensorfusionen utgör också en möjlig felkälla då projektet använde en förenklad fusionsmetod med medelvärden och enklare filtrering. Detta kan ge en mindre exakt uppskattning av objektets position än exempelvis Kalmanfilter. Djupinformationen från kamerorna var även syntetisk gjord från gråskalevärden, vilket inte ger samma noggrannhet som verkliga RGB-D-kameror. Koordinatransformationen mellan Unity och ROS 2 kan också påverka objekts position i den gemensamma världsbilden.

Unity kan även ha påverkat resultatet. Som nämnt i kapitel 2.2, har Unity en enklare inlärningskurva och mindre hårdvarukrav än UE5, vilket även kan begränsa prestanda och grafisk realism. Under projektets gång förekom det att Unity ofta kraschade, vilket kan ha lett till att ändringar inte sparats samt att tid behövde läggas på att återskapa eller kontrollera arbete. Projektets framsteg kan därför saktats ner och begränsat vidareutveckling. Användningen av tredjepartsplugins för 3DGS-rendering och kommunikation med ROS 2 kan även ha introducerat osäkerheter angående stabilitet och kompatibilitet.

8.5 Säkerhetsavvägningar och etiska aspekter

Eftersom projektet behandlar samarbete mellan människa och cobot är säkerhetsaspekten mycket viktig. Oavsett hur välimplementerat ett system är finns det alltid en risk för kollisioner eller felaktiga rörelser, särskilt om perceptionssystem eller banplanering inte fungerar korrekt. Därför är det viktigt att robotsystemet kan identifiera människor och andra objekt för att kunna anpassa sina rörelser. I projektet hanterades detta genom att definiera säkerhetszoner som gör att coboten avbryter eller omplanerar sin bana om externa objekt identifieras i området. Lösningen är däremot inte tillräckligt robust för säker användning utan kräver ytterligare utveckling och testning.

Eftersom projektet bygger på kamerabaserad perception och sensorfusion kan felaktiga segmenteringar och sensoravvikelser påverka systemets uppfattning av omgivningen, vilket i sin tur kan leda till felaktiga beslut i banplaneringen. För att systemet ska vara säkert är det därför starkt beroende av tillförlitlighet i perceptionssystemet och hur väl olika data kan sammanföras.

Användning av kameror och spårning i arbetsmiljön innebär även att försiktighet bör tas gällande integritet, eftersom människor kan identifieras och analyseras av

systemet. Vid implementation av dessa typer av system behöver därför hantering av sensordata och personrelaterad information följa relevanta riktlinjer och integritetskrav.

9
Slutsats

9.1 Slutsats av resultat

Resultatet av arbetet visar att 3DGS kan användas för att skapa en skalenlig och visuellt realistisk digital representation av en industriell miljö. Modellen kunde integreras i Unity och användas som grund för en digital tvilling av RITA-cellen. Detta visar att 3DGS är väl lämpat för att snabbt skapa fotorealistiska simuleringsmiljöer, särskilt när syftet är visualisering och systemintegration snarare än exakt fysikalisk simulering.

Den framtagna miljön visar även potential som plattform för utveckling och testning av robotapplikationer. Genom att kombinera 3DGS-modellen med cobotstyrning, segmentering, sensorfusion och VR kan flera centrala delar av en simulerad människa-robot-interaktion representeras i samma miljö. Detta innebär att digitala tvillingar av denna typ kan användas för att testa arbetsflöden, visualisera robotrörelser och vidareutveckla perceptions- och styrningsfunktioner innan implementation i verklig miljö.

Samtidigt visar arbetet att 3DGS har tydliga begränsningar för fysisk simulering. Eftersom 3DGS-modellen inte innehåller direkt användbar kollisionsgeometri behöver den kompletteras med CAD-modeller, förenklade kollisionsvolymer eller annan mesh-geometri för att kunna användas för tillförlitlig kollisionsdetektering och fysikbaserad interaktion. Detta begränsar modellens användbarhet för fullständig simulering av robotens kontakt med omgivningen.

Arbetet visar också att systemet bör betraktas som ett proof-of-concept. Flera delsystem implementerades och testades separat, men styrning och sensorfusion integrerades inte fullt ut i den slutliga simuleringsmiljön. Den slutliga miljön demonstrerar därför främst potentialen hos en integrerad digital tvilling, snarare än ett färdigt och validerat system för industriell användning.

Sammanfattningsvis besvaras frågeställningarna genom att 3DGS bedöms vara en användbar metod för att skapa visuellt realistiska digitala tvillingar av industriella miljöer, men mindre lämpad som ensam grund för fysikalisk simulering. Den digitala tvillingen är användbar för visualisering, integration och vidareutveckling av robotapplikationer, men kräver fortsatt arbete innan den kan användas för robust simulering, säkerhetskritisk testning eller minskning av sim-to-real gapet i praktiken.

9.2 Framtida arbete

För framtida arbete bör de separata delsystemen kopplas samman i en gemensam och stabil simuleringsmiljö. Detta omfattar framför allt fullständig integration av cobotstyrning, sensorfusion och segmentering i den slutliga Unity-miljön. En sådan integration skulle göra det möjligt att utvärdera hela systemkedjan från perception till banplanering och rörelse i en gemensam digital tvilling.

En viktig vidareutveckling är även att komplettera 3DGS-modellen med användbar kollisionsgeometri. Detta kan exempelvis göras genom CAD-modeller, förenklade kollisionsvolymer eller mer detaljerad mesh-generering från punktmoln eller 3DGS-data. Med förbättrad kollisionsgeometri skulle miljön kunna användas för mer realistisk fysiksimulering, säkrare banplanering och bättre utvärdering av interaktion mellan robot och omgivning.

Vidare bör systemet testas med fler scenarier, fler komponenttyper och mer varierande miljöförhållanden. Detta skulle ge bättre underlag för att utvärdera segmenteringens robusthet, sensorfusionens noggrannhet och robotstyrningens tillförlitlighet. På längre sikt kan miljön även användas för att generera träningsdata och undersöka hur väl modeller eller styrstrategier som utvecklas i simulering fungerar vid överföring till en verklig robotcell.

Referenser

[1] Itziar Ricondo, Alain Porto och Miriam Ugarte. ”A Digital Twin Framework for the Simulation and Optimization of Production Systems”. I: *Procedia CIRP* 104 (2021), s. 762–767. DOI: 10.1016/j.procir.2021.11.128. URL: https://www.sciencedirect.com/science/article/pii/S221282712101026X.

[2] McKinsey & Company. *Digital Twins: The Next Frontier of Factory Optimization*. URL: https://www.mckinsey.com/capabilities/operations/our-insights/digital-twins-the-next-frontier-of-factory-optimization (hämtad 2026-05-12).

[3] Inaki Maurtua m. fl. ”Human-Robot Collaboration in Industrial Applications: Safety, Interaction and Trust”. I: *International Journal of Advanced Robotic Systems* 14.4 (2017). DOI: 10.1177/1729881417716010. URL: https://journals.sagepub.com/doi/10.1177/1729881417716010.

[4] Alexandre Angleraud m. fl. ”Sensor-Based Human-Robot Collaboration for Industrial Tasks”. I: *Robotics and Computer-Integrated Manufacturing* 86 (2024), s. 102663. DOI: 10.1016/j.rcim.2023.102663. URL: https://www.sciencedirect.com/science/article/pii/S0736584523001382.

[5] Elie Aljalbout m. fl. ”The Reality Gap in Robotics: Challenges, Solutions, and Best Practices”. I: *Annual Review of Control, Robotics, and Autonomous Systems* (2026). DOI: 10.1146/annurev-control-031924-100130. URL: https://www.annualreviews.org/content/journals/10.1146/annurev-control-031924-100130.

[6] Bernhard Kerbl m. fl. ”3D Gaussian Splatting for Real-Time Radiance Field Rendering”. I: *ACM Transactions on Graphics* 42.4 (2023). DOI: 10.1145/3592433. URL: https://arxiv.org/abs/2308.04079.

[7] Santiago Berrezueta-Guzman och Stefan Wagner. ”Choosing the Right Engine in the Virtual Reality Landscape”. I: *IEEE Access* 14 (2026), s. 13972–13985. DOI: 10.1109/ACCESS.2026.3657272. URL: https://doi.org/10.1109/ACCESS.2026.3657272.

[8] *Merits and Demerits of Unreal and Unity: A Comprehensive Comparison*. URL: https://ieeexplore.ieee.org/document/10717602 (hämtad 2026-05-11).

[9] J. C. Feddema och L. Z. F. Chiu. ”Accuracy and Repeatability of 3D Photogrammetry to Digitally Reconstruct Bones”. I: *Morphologie* 108.361 (2024), s. 100793. DOI: 10.1016/j.morpho.2024.100793.

[10] Ben Mildenhall m. fl. ”NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. I: *Computer Vision – ECCV 2020*. Springer, 2020, s. 405–421. DOI: 10.1007/978-3-030-58452-8_24.

[11] XGRIDS. *PortalCam*. https://www.xgrids.com/intl/portalcamp. Hämtad: 2026-05-13. 2025.
[12] PlayCanvas. *SuperSplat*. URL: https://superspl.at/ (hämtad 2026-05-12).
[13] Aras Panteleev. *UnityGaussianSplatting: Gaussian Splatting Playground in Unity*. URL: https://github.com/aras-p/UnityGaussianSplatting (hämtad 2026-05-12).
[14] Lewis A. G. Stuart m. fl. "3DGS-to-PC: 3D Gaussian Splatting to Dense Point Clouds". I: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2025, s. 3730–3739. URL: https://openaccess.thecvf.com/content/ICCV2025W/3D-VAST/html/Stuart_3DGS-to-PC_3D_Gaussian_Splatting_to_Dense_Point_Clouds_ICCVW_2025_paper.html.
[15] Open Robotics. *ROS: Robot Operating System*. URL: https://www.ros.org/ (hämtad 2026-05-06).
[16] Open Robotics. *Understanding ROS 2 Nodes*. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html (hämtad 2026-05-06).
[17] Open Robotics. *Understanding ROS 2 Topics*. URL: https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html (hämtad 2026-05-06).
[18] PickNik Robotics. *MoveIt Documentation*. URL: https://moveit.picknik.ai/ (hämtad 2026-05-06).
[19] Mrinal Kalakrishnan m. fl. "STOMP: Stochastic Trajectory Optimization for Motion Planning". I: *2011 IEEE International Conference on Robotics and Automation*. 2011, s. 4569–4574. DOI: 10.1109/ICRA.2011.5980280. URL: https://ieeexplore.ieee.org/abstract/document/5980280.
[20] Matt Zucker m. fl. "CHOMP: Covariant Hamiltonian Optimization for Motion Planning". I: *The International Journal of Robotics Research* 32.9-10 (2013), s. 1164–1193. DOI: 10.1177/0278364913488805. URL: https://journals.sagepub.com/doi/10.1177/0278364913488805.
[21] Unity Technologies. *ROS-TCP Connector*. URL: https://github.com/Unity-Technologies/ROS-TCP-Connector (hämtad 2026-05-08).
[22] Joseph Redmon och Ali Farhadi. "YOLO9000: Better, Faster, Stronger". I: *2017 IEEE Conference on Computer Vision and Pattern Recognition*. 2017, s. 6517–6525. DOI: 10.1109/CVPR.2017.690.
[23] Juan Terven och Diana Cordova-Esparza. "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS". I: *Machine Learning and Knowledge Extraction* 5.4 (2023), s. 1680–1716. DOI: 10.3390/make5040083.
[24] Alexander Kirillov m. fl. "Segment Anything". I: *arXiv preprint arXiv:2304.02643* (2023). URL: https://arxiv.org/abs/2304.02643.
[25] Xu Zhao m. fl. "Fast Segment Anything". I: *arXiv preprint arXiv:2306.12156* (2023). URL: https://arxiv.org/abs/2306.12156.
[26] Enze Xie m. fl. "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers". I: *Advances in Neural Information Processing Systems*. Vol. 34. 2021. URL: https://proceedings.neurips.cc/paper/2021/hash/64f1f27bf1b4ec22924fd0acb550c235-Abstract.html.

[27] David L. Hall och James Llinas. ”An Introduction to Multisensor Data Fusion”. I: *Proceedings of the IEEE* 85.1 (1997), s. 6–23. DOI: 10.1109/5.554205. URL: https://ieeexplore.ieee.org/document/554205.

[28] Spyros G. Tzafestas et al. *Mobile Robot Localization and Mapping*. 2014. DOI: 10.1016/C2013-0-01365-5. URL: https://www.sciencedirect.com/science/chapter/monograph/pii/B9780124170490000122.

[29] Sandra Robla m. fl. ”Visual Sensor Fusion for Active Security in Robotic Industrial Environments”. I: *EURASIP Journal on Advances in Signal Processing* 2014.1 (2014). DOI: 10.1186/1687-6180-2014-88. URL: https://link.springer.com/article/10.1186/1687-6180-2014-88.

[30] *Novel Gray Scale Conversion Techniques Based on Pixel Depth*. URL: https://www.rroij.com/open-access/novel-gray-scale-conversion-techniques-based-on-pixel-depth.php?aid=37541 (hämtad 2026-05-10).

[31] Martin Ester m. fl. ”A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. I: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. 1996, s. 226–231. URL: https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf.

[32] Stanislav Svediroh och Ludek Zalud. ”Atlas Fusion 2.0: A ROS2-Based Real-Time Sensor Fusion Framework”. I: *Computer Vision Systems*. Springer, 2024, s. 25–37. DOI: 10.1007/978-3-031-71397-2_3. URL: https://link.springer.com/chapter/10.1007/978-3-031-71397-2_3.

[33] Nicolai Wojke, Alex Bewley och Dietrich Paulus. ”Simple Online and Realtime Tracking with a Deep Association Metric”. I: *2017 IEEE International Conference on Image Processing*. 2017, s. 3645–3649. DOI: 10.1109/ICIP.2017.8296962. URL: https://ieeexplore.ieee.org/document/8296962.

[34] H. Didem Uzgun och Emre Koyuncu. ”Dynamic Confidence Weighted Visual-Inertial Sensor Fusion for Global Positioning System-Denied Navigation”. I: *Engineered Science* 40 (2026), s. 2065. DOI: 10.30919/es2065. URL: https://research.itu.edu.tr/en/publications/dynamic-confidence-weighted-visual-inertial-sensor-fusion-for-glo/.

[35] Albin Dahlin. ”Reactive Motion Planning and Control under Constraints”. Diss. Chalmers University of Technology, 2023. URL: https://research.chalmers.se/publication/538686/file/538686_Fulltext.pdf (hämtad 2026-05-12).

[36] Federica Ferraguti m. fl. ”Safety Barrier Functions and Multi-Camera Tracking for Human-Robot Shared Environment”. I: *Robotics and Autonomous Systems* 124 (2020), s. 103388. DOI: 10.1016/j.robot.2019.103388. URL: https://www.sciencedirect.com/science/article/abs/pii/S0921889019306426.

[37] *How To Make A VR Game in an Hour Using Unity 6!* URL: https://www.youtube.com/watch?v=kbBYcVrGZus (hämtad 2026-05-06).

[38] Unity Technologies. *VR and MR Development in Unity*. URL: https://docs.unity3d.com/Manual/VROverview.html (hämtad 2026-05-06).

[39] Unity Technologies. *XR Packages*. URL: https://docs.unity3d.com/Manual/xr-support-packages.html#plug-ins (hämtad 2026-05-06).

[40] *SteamVR*. URL: https://vrarwiki.com/wiki/SteamVR (hämtad 2026-05-06).

[41] HTC Corporation. *VIVE Hub*. URL: https://store.steampowered.com/app/1649260/VIVE_Hub/ (hämtad 2026-05-06).

[42] HTC Corporation. *VIVE Hub: The Magic of Lossless PCVR Streaming, Spatial Workspaces, and Full-Body Tracking Setup*. URL: https://blog.vive.com/us/vive-hub-the-magic-of-lossless-pcvr-streaming-spatial-workspaces-and-full-body-tracking-setup/ (hämtad 2026-05-06).

[43] Adobe. *Mixamo Dummy Avatar*. URL: https://www.mixamo.com/#/?page=2&type=Character (hämtad 2026-05-08).

[44] *Complete VR Body Setup: Arms and Legs IK with Hand Animation*. URL: https://www.youtube.com/watch?v=v47lmqfrQ9s (hämtad 2026-05-06).

[45] *Inverse Kinematics Explained*. URL: https://sandboxvr.com/articles/inverse-kinematics (hämtad 2026-05-10).

[46] Unity Technologies. *Animation Rigging*. URL: https://docs.unity3d.com/Packages/com.unity.animation.rigging@1.0/manual/index.html (hämtad 2026-05-10).

[47] *VR Full Body*. URL: https://drive.google.com/file/d/1eSaeMTLxWpRYo8ZYM6hprlMG-N716REL/view (hämtad 2026-05-10).

A

Appendix 1

A.1 Google Drive länk

Google drive folder som innehåller demovideo av resultatet där en kittingsekvens utförs och två videor av styrningen.

`https://drive.google.com/drive/folders/1-lybuCF1dvITY6yMjbpsbi2BGJvG1Gyk?usp=sharing`

A.2 Kod och repository segmentering

Koden som användes för segmenteringsdelen finns tillgänglig i detta GitHub-repository:

`https://github.com/callewallerstedt/KandidatarbeteCoBot`

Repositoryt innehåller kod för att skapa syntetisk träningsdata, träna segmenteringsmodellen samt köra inferens och tester.

A.3 Kod och repository för styrning

Kod som använts inom projektet finns tillgänglig i ett GitHub-repository. Repositoryt innehåller implementationer kopplade till kommunikationen mellan Unity och ROS 2, styrning av den simulerade robotmodellen samt övriga scripts som använts för integrationen i Unity-miljön.

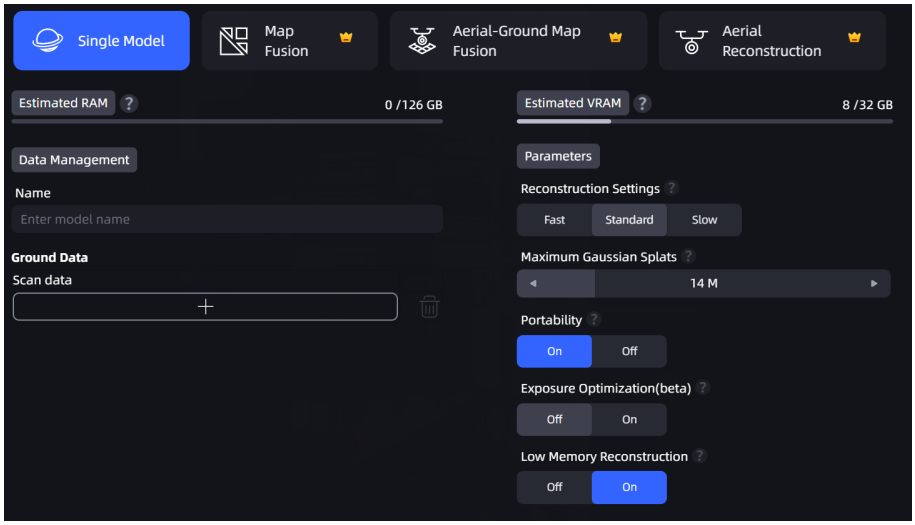
Repository: `https://github.com/erichmberg/unity_ros2`

A.4 Kod och repository för sensorfusion

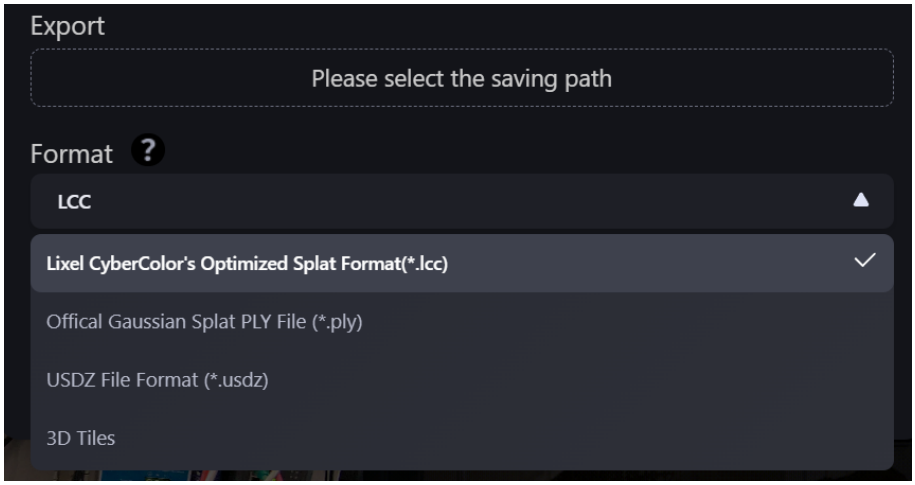
Kod som använts inom projektet finns tillgänglig i ett GitHub-repository. Repositoryt kopplar till kommunikationen mellan Unity och ROS 2, sammansättningen av dataflöden samt scripts för säkerhet.

Repository: `https://github.com/erichmberg/Sensor-Fusion`

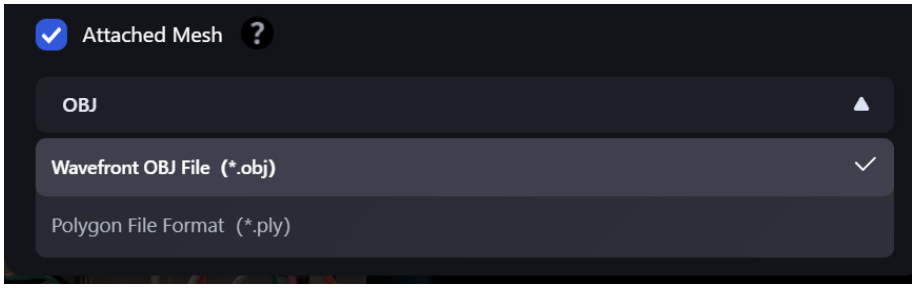
A.5 Inställningar för 3DGS-bearbetning i LCC Studio



Figur A.1: Inställningar för bearbetning av 3DGS modeller i LCC Studio.



Figur A.2: Exportalternativ av bearbetad 3DGS-modell i LCC Studio.



Figur A.3: Exportalternativ för mesh i LCC Studio.

Institutionen för Elektroteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS