CHALMERS

Realistisk träningsdata med 3D Gaussian Splatting

Bygga digitala tvillingar för generering av ground-truth-data i Unreal Engine 5

Kandidatarbete inom Elektroteknik

HUSSEIN AHMED, LINNEA BODIN, EMIL JANSSON, LUCAS NILSSON, MELKER SJÖSTEDT, DAVID HAFSTAD WALLANDER

Institutionen för Elektroteknik

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2025
www.chalmers.se

Kandidatarbete 2025

Realistisk träningsdata med 3D Gaussian Splatting

Bygga digitala tvillingar för generering av ground-truth-data i Unreal Engine 5

HUSSEIN AHMED
LINNEA BODIN
EMIL JANSSON
LUCAS NILSSON
MELKER SJÖSTEDT
DAVID HAFSTAD WALLANDER

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2025

Realistisk träningsdata med 3D Gaussian Splatting
Bygga digitala tvillingar för generering av ground-truth-data i Unreal Engine 5
HUSSEIN AHMED, LINNEA BODIN, EMIL JANSSON, LUCAS NILSSON,
MELKER SJÖSTEDT, DAVID HAFSTAD WALLANDER

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

Kandidatarbete 2025
Institutionen för Elektroteknik
EENX16-VT25-50
Chalmers Tekniska Högskola
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Omslagsbild: Visar en modell av en autonom robot skapad med 3D Gaussian Splatting.

Skriven i LaTeX
Göteborg, Sverige 2025

Abstract

Modern industrial environments are becoming increasingly complex and automated, increasing the need for autonomous robots for internal logistics. For these robots to work efficiently and safely, high-quality, domain-specific training data is required. The production of such data has traditionally been a resource-intensive process.

This bachelor thesis evaluates, as a proof of concept, the potential of 3D Gaussian Splatting (3DGS) to create photorealistic digital twins of industrial environments. The aim was to automatically generate synthetic ground truth data from these twins.

The methodology involved photographing real-world industrial environments (AB Volvo's factory in Tuve, Chalmers Production System Laboratory) and creating 3D Gaussian Splatting models. After importing these models into Unreal Engine 5 (UE5), functionality was developed to simulate scenarios and automatically generate image pairs, each consisting of a simulated view and a corresponding segmentation map for semantic segmentation.

The results indicated that 3D Gaussian Splatting can produce detailed digital twins, despite some quality loss during import into Unreal Engine 5. The automatic data generation proved effective, producing synthetic data that enabled the training of a segmentation model. The model was able to classify objects in real factory images, albeit with certain identified limitations.

The conclusion is that 3DGS is a promising technique to streamline generation of training data. Despite challenges with image capture and lack of support for 3DGS in Unreal Engine 5, the work shows that the investigated method is promising to facilitate the development of advanced machine learning models.

Keywords: 3D Gaussian Splatting, Synthetic Training Data, Ground-Truth, Digital Twins, Simulation, Unreal Engine 5, Semantic Segmentation, Data Generation, Factory Environments.

Sammandrag

Moderna industrimiljöer blir alltmer komplexa och automatiserade, vilket ökar behovet av autonoma robotar för intern logistik. För att dessa robotar ska kunna arbeta effektivt och säkert krävs högkvalitativ träningsdata. Framtagningen av sådan data har traditionellt sett varit en resurskrävande process.

Detta kandidatarbete utvärderar, som ett 'proof-of-concept', potentialen hos 3D Gaussian Splatting (3DGS) för att skapa fotorealistiska digitala tvillingar av industriella miljöer. Syftet är att från dessa tvillingar automatiskt generera syntetisk ground-truth-data.

Metoden innefattade fotografering av verkliga industrimiljöer (AB Volvos fabrik i Tuve, Chalmers produktionssystemslabbet) och skapande av 3D Gaussian Splatting modeller. Efter importering till Unreal Engine 5 utvecklades funktionalitet som simulerade scenarion och automatiskt genererade bildpar bestående av en simulerad vy och en motsvarande segmenteringskarta för semantisk segmentering.

Resultaten visade att 3D Gaussian Splatting kan producera detaljerade digitala tvillingar, trots viss kvalitetsförlust vid import till Unreal Engine 5. Den automatiska datagenereringen var effektiv och producerade syntetisk data som möjliggjorde träning av en segmenteringsmodell. Modellen kunde klassificera objekt i verkliga fabriksbilder, om än med vissa identifierade begränsningar.

Slutsatsen är att 3DGS är en lovande teknik för att effektivisera framtagning av träningsdata. Trots utmaningar med bildtagning och bristande stöd för 3DGS i Unreal Engine 5 visar arbetet att den undersökta metoden är lovande för att underlätta utvecklingen av avancerade maskinlärningsmodeller.

Nyckelord: 3D Gaussian Splatting, Syntetisk Träningsdata, Ground-Truth, Digitala Tvillingar, Simulering, Unreal Engine 5, Semantisk Segmentering, Datagenerering, Industriella Miljöer.

Tillkännagivande

Vi vill inleda med att rikta ett stort tack till Knut Åkesson och Kristian Ceder för deras ovärderliga stöd och vägledning under hela projektets gång. Deras insikter har varit en stor hjälp för oss.

Vi vill även uttrycka vår uppskattning till AB Volvo, och Kristofer Bengtsson, vars hjälp har bidragit till att utveckla och förbättra vårt arbete. Slutligen vill vi tacka vår examinator, Martin Fabian, för hans återkoppling och vägledning.

Hussein Ahmed, Linnea Bodin, Emil Jansson, Lucas Nilsson,
Melker Sjöstedt, David Hafstad Wallander, Göteborg, Mars 2025

Akronymer

Nedan följer en lista över akronymer som har använts i detta examensarbete, listade i alfabetisk ordning:

3DGS	3D Gaussian Splatting
AI	Artificiell Intelligens
IoU	Intersection over Union
MRQ	Movie Render Queue
NeRF	Neural Radiance Fields
PLY	Polygon File Format
POC	Proof Of Concept
PSL	Productionssystemlabbet
RGB	Red Green Blue
SfM	Structure from Motion
UE5	Unreal Engine 5

Innehåll

Figurer

1
Introduktion

Industrimiljöer blir alltmer automatiserade, vilket ställer höga krav på autonoma robotars förmåga att navigera och interagera effektivt med sin omgivning. Detta projekt undersöker användningen av 3D Gaussian Splatting (3DGS) för att skapa visuellt realistiska digitala tvillingar av fabriksmiljöer. Dessa verklighetstrogna digitala tvillingar kan sedan användas för att generera träningsdata för maskininlärningsmodeller avsedda att styra de autonoma robotarna. Syftet är att utvärdera om 3DGS är en lämplig metod för att generera denna träningsdata och därmed underlätta utvecklingen av autonoma system. Denna rapport presenterar processen för att skapa en digital tvilling, generera ground-truth-data och utvärdera metodens potential.

1.1 Bakgrund

Dagens samhälle genomsyras i hög grad av teknologins framfart. Genom nya innovationer och vidareutveckling av befintliga lösningar har digitaliseringen effektiviserat verktyg, tjänster och arbetsprocesser inom flera sektorer [1]. Två centrala tekniker i denna utveckling är automatisering och artificiell intelligens. Teknikerna har väckt stort intresse hos AB Volvo, som står inför en komplex logistisk utmaning [2].

Vid AB Volvos fabrik i Tuve tillverkas flera olika lastbilsmodeller på en enda produktionslinje. I dagsläget hanteras omkring 250 000 unika delar per dag, en siffra som Volvo förväntar sig kommer att öka avsevärt de kommande åren. För att säkerställa att rätt delar finns på rätt plats vid rätt tidpunkt har AB Volvo undersökt möjligheten att använda självkörande, autonoma robotar för intern transport och logistik [2]. Sådana robotar är redan i drift inom delar av produktionen och har introducerats framgångsrikt. Eftersom även AstraZeneca visade intresse för att integrera autonoma robotar i sin verksamhet, inleddes ett gemensamt forskningsprojekt. Samarbetet omfattar AB Volvo, AstraZeneca och Chalmers tekniska högskola och fokuserar på design och kontroll av mobila, kollaborativa robotar.

Vid utvecklingen och integreringen av robotarna hos AB Volvo identifierade de ett antal utmaningar, främst på grund av fabriksmiljöns komplexitet. En central fråga var hur styrsystemet ska designas för att robotarna ska kunna anpassa sig till olika händelser, såsom mötande fordon och blockerade vägar. Utvecklarna fick idén att använda kameror i fabrikens tak som en del av robotarnas perceptionssystem. Därefter kan en maskininlärningsmodell tränas för att detektera och klassificera objekt

i miljön, vilket i sin tur kan nyttjas för att förbättra robotarnas styrning.

För att en sådan modell ska uppnå hög precision i fabriksmiljöer krävs en omfattande mängd domänspecifik träningsdata. Annoteringen av denna data har traditionellt skett manuellt, en process som är resurs- och tidskrävande samt svår att skala upp. För att maskininlärningsmodellen ska bli robust krävs även att man simulerar hur roboten beter sig i ett stort antal potentiella situationer. Sådana simuleringar erbjuder en säker och flexibel miljö för att testa och validera modellen under kontrollerade förhållanden, utan avbrott i produktionen

De identifierade problemen ledde till förslaget att skapa en digital tvilling av fabriksmiljön. Målet med projektet är således att undersöka möjligheten att använda 3D Gaussian splatting (3DGS) för att skapa en digital tvilling, driven av idén om att skapa en sammanhängande och verklighetstrogen virtuell återgivning av verkligheten. Med en sådan modell kan annotering av data göras automatiskt och flera scenarier kan simuleras.

1.2 Syfte

Syftet med detta projekt är att utveckla och utvärdera en metod för att generera annoterad träningsdata för maskininlärningsmodeller. Detta uppnås genom att skapa en realistisk digital tvilling av en industriell miljö med hjälp av 3D Gaussian Splatting (3DGS).

1.3 Avgränsningar

De digitala tvillingarna som skapas i projektet kommer att baseras på två specifika verkliga platser: produktionssystemslabbet på Chalmers och en avgränsad del av produktionsmiljön hos AB Volvo, närmare bestämt det område där deras autonoma robotar för närvarande opererar.

Den semantiska segmenteringen i den genererade datan avgränsas till tre grundläggande objektklasser som är centrala för robotnavigation: robot, golv (körbar yta) och hinder. Klassen ”hinder” definieras här funktionellt och innefattar alla ytor eller objekt som inte klassificeras som golv och därmed utgör icke-körbara områden för roboten. En mer detaljerad klassificering av olika typer av hinder ligger utanför detta arbetes omfattning.

Projektet inkluderar inte framtagning eller användning av ett manuellt annoterat dataset från de verkliga miljöerna för kvantitativ jämförelse. Utvärderingen av den tränade modellens prestanda baseras därför på kvalitativa observationer av dess förmåga att segmentera verkliga bilder samt på analys av den syntetiska datagenereringsprocessen, snarare än på formella noggrannhetsmått beräknade mot en verklig ground-truth-datamängd.

1.4 Problem

Utvecklingen och driftsättningen av robusta autonoma system i komplexa industriella miljöer är starkt beroende av tillgång till stora mängder högkvalitativ, annoterad träningsdata. Denna data är nödvändig för att träna de maskininlärningsmodeller som utgör kärnan i robotarnas perceptions- och beslutssystem. Nuvarande metoder för insamling och manuell annotering av sådan data är oftast tidskrävande, kostsam samt svårt att skala upp. Dessa begränsningar utgör en flaskhals som hämmar utvecklingstakten och försvårar en bredare implementering av autonoma robotar.

Följaktligen existerar ett tydligt behov att utforska och utvärdera alternativa, mer effektiva, och skalbara metoder för att generera nödvändig träningsdata. Frågeställningen är huruvida 3D Gaussian Splatting är en lämplig metod för att skapa fotorealistiska digitala representationer, som sedan kan användas för att effektivt generera syntetisk ground-truth-data.

2
Digitala tvillingar

En digital tvilling är en digital representation av ett verkligt system [3]. Systemet kan vara av ett specifikt objekt, en uppsättning komponenter eller ett utrymme. Målet är att skapa en replika av verkligheten och därigenom kunna simulera systemet. Genom att tillföra realtidsinformation kan även systemets framtid förutsägas, vilket bland annat kan användas i styralgoritmer för autonoma robotar. Detta kapitel introducerar användningsområden för digitala tvillingar, samt en beskrivning av hur de kan byggas upp.

2.1 Användningsområden för digitala tvillingar

I dagsläget finns det tre huvudområden där digitala tvillingar appliceras [4]. Dessa är inom produktdesign, produktion samt området prognostik och hälsotillståndshantering. Inom Designprocessen kan effektiviseras och göras mer informationsdriven genom att använda data från en digital tvilling, vilket möjliggör tidigare och mer träffsäkra utvärderingar av produkten jämfört med traditionella metoder. Digitala tvillingar kan också öka synkronisering mellan design och produktion, eftersom produktionsplaneringen kan påbörjas parallellt med designarbetet. Detta skapar ett samspel där designbeslut kan fattas med bättre förståelse för hur produkten interagerar med sin produktionsmiljö. Produktionsprocesser kan övervakas med hjälp av digitala tvillingar genom visualisering av realtidssystem. Med hjälp av visualiseringen kan operatörer korrigera och optimera komplexa tillverkningsflöden. Genom att integrera data från produktionsprocesser kan justeringar även utföras autonomt under pågående operationer. Förutom autonom optimering av produktionen kan justeringarna reducera materialsvinn samt förlänga tillverkningsmaskiners livslängd.

Utifrån data har digitala tvillingar möjlighet att underlätta underhåll av system genom att analysera operationstillstånd, diagnostisera eventuella fel och rekommendera åtgärder [4]. Detta kallas för prognostik och hälsotillståndshantering och appliceras på system där driftstörningar riskerar att orsaka allvarliga ekonomiska eller säkerhetsrelaterade följder. Digitala tvillingar användes först i flygplan för detta syfte men har sedan dess spritt sig till produktionssystem och andra fordon [4]. Utöver att rapportera aktuella hälsotillstånd kan även ett systems återstående livslängd och sannolikhet för systemfel förutsägas. Genom digitala tvillingar kan potentiella risker identifieras och därigenom bidra till att förebygga olyckor.

2.2 Bygga digitala modeller av fabriker

Enligt Tao et al. [4] finns det olika uppfattningar kring vad en digital tvilling består av, men de föreslår en generell modell med fem dimensioner. Modellen består av det fysiska, det virtuella, data, tjänster och sammankoppling. Den fysiska delen utgör grunden för tvillingen och representerar det verkliga system som ska modelleras. Den virtuella delen är dess digitala motsvarighet och möjliggör simulering, beslutsfattande och styrning av det fysiska systemet. Data står i centrum för digitala tvillingar och fungerar som bas för beslut och styrning. Tjänster är olika användarverktyg som stödjer funktionalitet och bidrar till ökad tillförlitlighet och effektivitet. För att dessa komponenter ska fungera som ett sammanhängande system krävs en sammankoppling som säkerställer ett kontinuerligt flöde av data och information mellan delarna.

Övergången från ett fysiskt system till en virtuell modell kan variera beroende på användningsområdet. Tao och Zhang [5] utvecklade en digital tvilling av ett produktionsgolv genom att skapa tredimensionella modeller för fysiska objekt med hjälp av program som SolidWorks, AutoCAD och CATIA. I ett produktionssystem, där den visuella noggrannheten inte är i fokus, fungerar dessa geometrier bra för att representera objektens former och positioner. Om syftet med den digitala tvillingen kräver högre visuell noggrannhet finns det flera metoder som gör detta möjligt. Några av dessa presenteras i kommande avsnitt (2.2.1 till 2.2.4).

2.2.1 Laserskanning och punktmoln

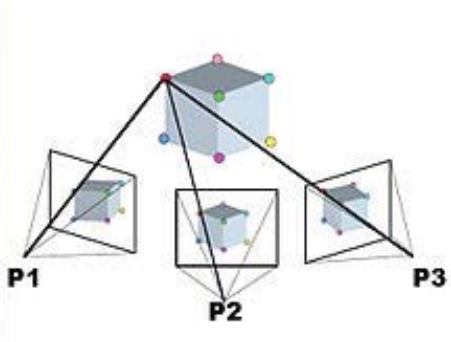
Laserskanning är en metod som använder laserljus för att fånga storleken och formen av ett föremål eller en miljö. Lasern lyser upp objektet medan en sensor läser av det reflekterade ljuset från dess yta [6]. Skannerns programvara beräknar ett antal avståndsvärden, som därefter kan konverteras till punkter i ett 3D-koordinatsystem. Punkterna utgör ett punktmoln; en tredimensionell uppsättning datapunkter som representerar ett objekts yta [7]. För ett exempel, se figur 2.1.

Figur 2.1: Exempel på ett punktmoln av Monmouth slott. [8], CC-BY-SA

2.2.2 Structure from Motion

Structure from Motion (SfM) är en fotogrammetrisk metod som likt laserskanning skapar ett resultat i form av ett punktmoln. Skillnaden ligger i att SfM utgår från en samling överlappande bilder på det objekt som man vill avbilda [9]. Dessa bildsamlingar kan antingen fångas genom en video där specifika bildrutor väljs ut eller genom vanlig fotografering av objektet. Från varje bild identifieras nyckelpunkter [9], där antalet nyckelpunkter per bild beror på bildens upplösning samt komplexitet. Högupplösta bilder med fler detaljer resulterar därför vanligtvis i fler nyckelpunkter.

Nyckelpunkter som uppstår i flera bilder matchas och spår, som länkar specifika nyckelpunkter i en bildserie, etableras [9]. Spår som består av minst två nyckelpunkter och tre bilder används för att senare rekonstruera punktmolnet, medan spår som inte uppfyller kriterierna filtreras bort. Dessa matchade nyckelpunkter begränsar bildernas möjliga kameraorienteringar, vilket möjliggör uppskattning av kamerapositionerna i 3D-rummet. Med hjälp av triangulering kan nyckelpunkternas 3D-koordinater beräknas och ett punktmoln skapas, se figur 2.2. Ur ett punktmoln är det även möjligt att konstruera en 3D-mesh, se avsnitt 4.1.3



Figur 2.2: Exempel på ett spår som består av 3 bilder och 7 nyckelpunkter. Utifrån matchade nyckelpunkter kan bildernas kameraorientering estimeras och ett punktmoln skapas. [10], CC-BY-SA

2.2.3 Neural Radiance Field

Neural Radiance Field (NeRF) är en metod som rekonstruerar 3D-miljöer genom att träna ett neuronnät på en uppsättning bilder tillsammans med tillhörande kameraorienteringar [11] vilka vanligtvis erhålls genom en SfM-algoritm. Neuronnätet producerar ett strålningsfält, radiance field på engelska, där punkter i det tredimensionella rummet erhåller en färg och densitet för att representera miljön.

För varje bild med känd kameravinkel skickas en stråle genom scenen för varje pixel. Längs varje stråle provtas ett antal punkter i det tredimensionella rummet. Dessa punkter matas in i ett neuronnät, som för varje punkt försöker förutsäga vilken färg och opacitet den har från den aktuella synvinkeln. Genom att jämföra prediktionen med den verkliga bilden kan nätverket lära sig hur scenen är uppbyggd. Denna process upprepas flera gånger för samtliga bilder och tillhörande kameraorienteringar.

Till slut har neuronnätet lärt sig förutsäga färg och täthet, i varje 3D-punkt, givet en godtycklig synvinkel. Detta gör det möjligt att rendera miljön ur nya kameravinklar som inte fanns med i det ursprungliga bildmaterialet genom att kasta strålar från dessa nya positioner och låta neuronnätet returnera hur miljön bör renderas.

2.2.4 3D Gaussian Splatting

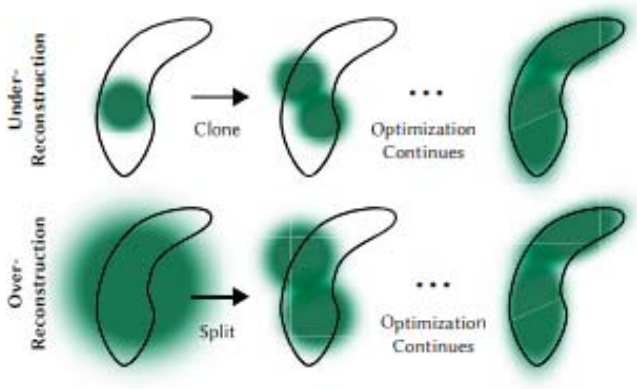
3D Gaussian splatting (3DGS), likt NeRF, är en metod som används för att återskapa 3D-miljöer från en samling överlappande bilder [12] och konstruera ett strålningsfält. Utifrån en SfM-algoritm, se avsnitt 2.2.2, erhålls ett punktmoln samt beräknade kameraorienteringar. Utifrån punktmolnet initieras en uppsättning 3D-gaussians, som definieras av en position, form och opacitet, samt en vyberoende färg. Utifrån dessa egenskaper representeras en 3D-gaussian som en ellipsoid med högst opacitet i centrum, där genomskinligheten avtar med avståndet från dess mittpunkt. Färgen är dessutom beroende av observationsvinkeln och kan därför variera beroende på betraktelseperspektiv. För exempel på hur enskilda 3D-gaussians kan se ut, se figur 2.3.

Figur 2.3: Grupp av 3D-gaussians som visar att alla har egen position, form, opacitet och färg.

Efter initiering av 3D-gaussians påbörjas en optimeringsfas där miljön iterativt renderas från samma kameravinklar som de ursprungliga bilderna [12]. Baserat på skillnaderna mellan renderingen och ursprungsbilderna uppdateras individuella egenskaper såsom position, form, färg och opacitet för varje 3D-gaussian med mål att få renderingen att efterlikna referensbilderna så nära som möjligt.

Vid var hundrade iteration hanteras antalet 3D-gaussians baserat på om miljön är under- eller överrekonstruerad [12], se figur 2.4. I underrekonstruerade områden saknas 3D-gaussians, vilket gör att scenen blir ofullständig. För att åtgärda detta klonas närliggande 3D-gaussians och inkluderas i optimeringsprocessen för att återställa mer detaljer. Överrekonstruktion inträffar när enskilda 3D-gaussians täcker

en för stor yta, vilket hindrar en noggrann återgivning av scenen. I sådana fall delas större 3D-gaussians upp i två mindre. På detta sätt ökar tätheten gradvis, vilket möjliggör en mer detaljerad rekonstruktion än den som ursprungligen genererades från punktmolnet. I båda fallen leder detta till att tätheten av 3D-gaussians ökar, vilket gör att fler detaljer kan rekonstrueras. Slutligen för var hundrade iteration tas praktiskt taget osynliga 3D-gaussians bort om deras opacitet understiger ett fördefinierat tröskelvärde.



Figur 2.4: Exempel på hur en 3D-Gaussian både klonar och delar sig under optimeringsprocessen [12].

2.3 Jämförelse och val mellan 3DGS och NeRF

Basso et al. [13] utförde en jämförelse mellan 3DGS och NeRF och kom fram till att NeRF var bättre på att hantera rekonstruktioner där bildvyer saknades. Däremot hade modeller skapade med 3DGS bättre texturer, mer realistisk återgivning av material samt bättre hantering av ljus och skuggor. Kerbl et al. [12] genomförde jämförelser mellan 3DGS och NeRF på tre dataset där 3DGS visade sig rendera mer realistiska bilder enligt Structural Similarity Index, ett mått på skillnader mellan en förvrängd bild och en referensbild [14]. 3DGS visade sig även vara mer effektiv än NeRF när det kommer till både rendering och träningstider. 3DGS nådde upp mot 154 bilder per sekund medan NeRF endast nådde 0,14 bilder per sekund. Träningstiderna för 3DGS-modellerna tog 26, 36 och 41 minuter beroende på dataset medan NeRF krävde 48 timmar på alla tre.

På grund av dess kapacitet att rendera i realtid, med många bilder per sekund, noggrannare rendering och snabbare träningstider valdes 3DGS som metod för att skapa den digitala tvillingen.

3

Modellbygge med 3D Gaussian Splatting

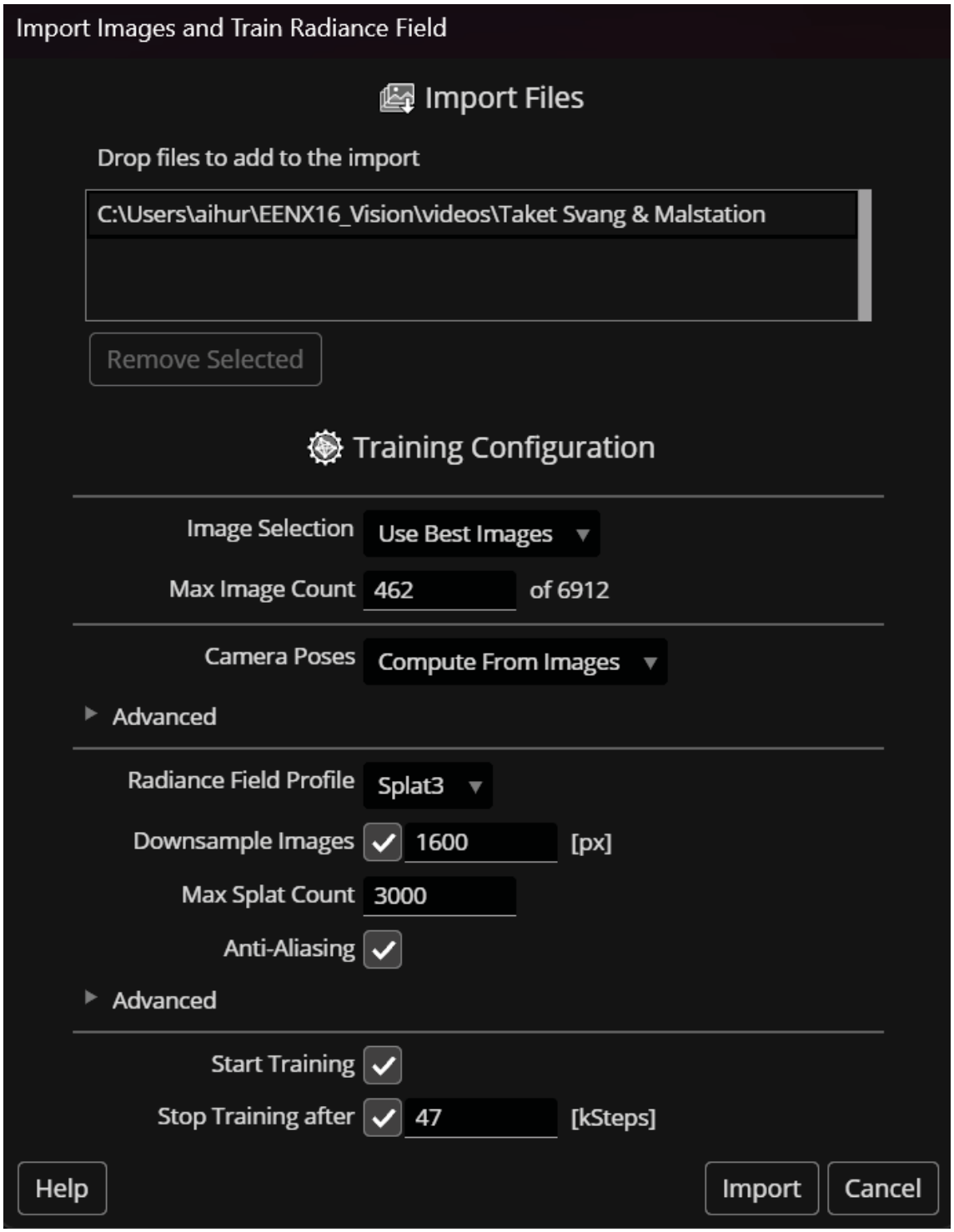
I detta kapitel redovisas arbetsprocessen för att skapa 3D-modeller med hjälp av ett mjukvaruverktyg baserat på tekniken 3D Gaussian Splatting (3DGS). Denna teknik möjliggör fotorealistisk återgivning av både verkliga miljöer och objekt. Kapitlet omfattar en teoretisk genomgång av mjukvaran samt en beskrivning av metodiken, från bildtagning till generering av en färdig 3D-modell. Avslutningsvis behandlas hur modellerna kan manipuleras samt hur de exporteras för vidare användning.

3.1 Mjukvaruprogram för 3D-modellering med 3D Gaussian Splatting

Postshot är ett mjukvaruprogram som bygger på metoden som presenterades i avsnitt 2.2.4, och används för att generera en 3DGS-modell utifrån en uppsättning bilder eller en video. Programmet, efter importering av bilder eller video, ger ett antal val i träningskonfigurationen. Efter påbörjas en trefasig process indelad i följande steg: bildbehandling (image processing), kameraspårning (camera tracking) samt träning av radiance field [15].

3.1.1 Träningskonfiguration

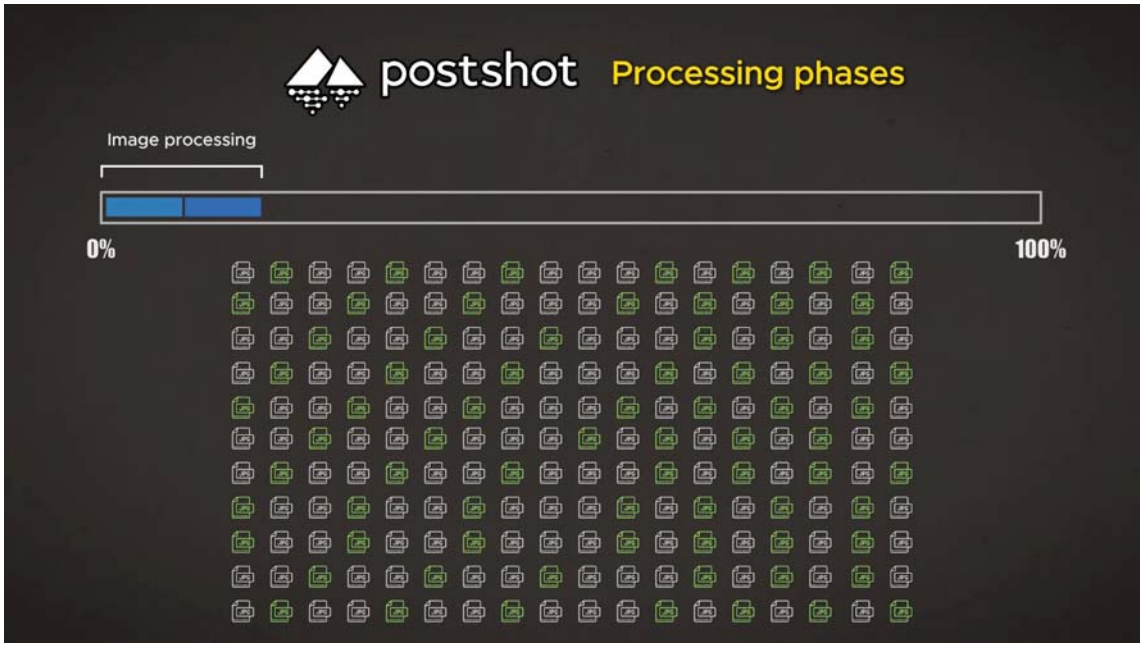
När en video eller ett antal bilder har importerats i Postshot, presenteras flera alternativ för träningskonfiguration. Om en video importeras, konverteras dess bildrutor automatiskt till en serie stillbilder. Figur 3.1 visar standardinställningarna som används under träningen. Postshot föreslår automatiskt ett lämpligt antal baserat på det totala antalet bilder, vilket kan ses vid *Max Image Count* i figur 3.1. Det går även att använda samtliga bilder genom att ändra alternativet *Image Selection: Use All Images* [16].



Figur 3.1: Träningskonfigurationen i Postshot efter att en video eller en uppsättning bilder har importerats.

3.1.2 Bildbehandling

Den första delen av träningsprocessen, bildbehandling, itererar över de importerade bilderna (se figur 3.2) för att identifiera de som är skarpa och väl utspridda över miljön för att möjliggöra effektiv kameraspårning. Detta steg utförs endast om alternativet *Image Selection: Use Best Images* är aktiverat. I annat fall går programmet direkt vidare till nästa steg, kameraspårning, se avsnitt 3.1.3 [16].

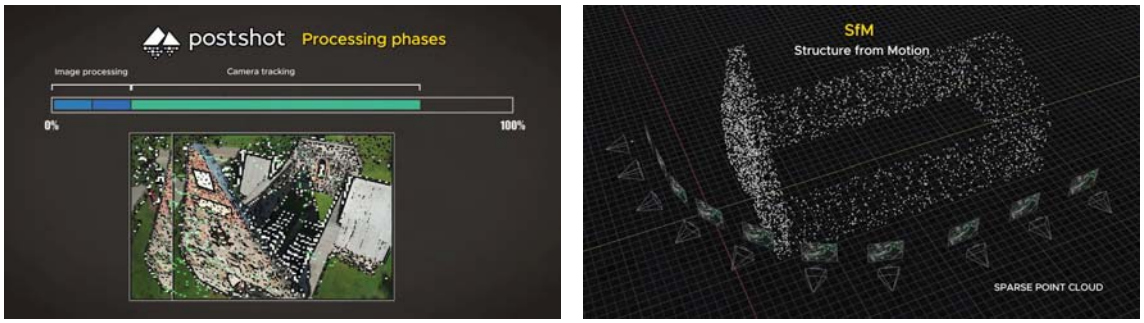


Figur 3.2: Bildbehandlingsprocessen i Postshot. Processfältet längst upp visar den tid som detta processteg generellt utgör i procent. De gröna ikonerna representerar de valda bilder. [17]. Återgiven med tillstånd.

.

3.1.3 Kameraspårning

Kameraspårningen är det processteg som tar längst tid, se det gröna området i förloppsindikatorn i figur 3.3 (a). Den bygger på det som beskrivs i avsnitt 2.2.2, där lämpliga nyckelpunkter identifieras i överlappande bilder, se figur 3.3 (a). Figur 3.3 (b) visar det rekonstruerade punktmolnet från nyckelpunkter som identifieras i flera bilder, skapat av Structure from Motion (SfM) metoden. [16].

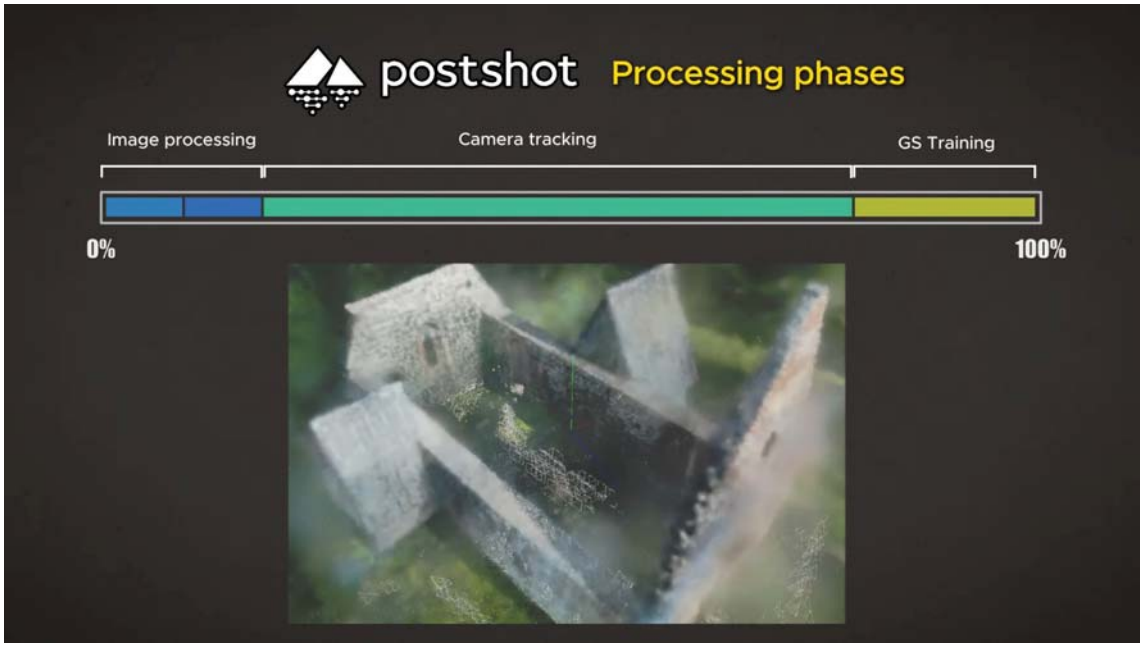


(a) Nyckelpunkter **(b)** Punktmoln

Figur 3.3: Processen för kameraspårning i Postshot. (a) Visar valda nyckelpunkter i bilderna samt den procentuella tiden för det aktuella steget och (b) visar det 3D punktmoln som genererats av Structure from Motion (SfM) metoden. [17]. Återgiven med tillstånd.

3.1.4 Träning av radiance field

Slutligen inleds det sista steget i processen, där träningen av 3D Gaussian Splatting påbörjas, vilket beskrivs mer ingående i avsnitt 2.2.4. Under detta steg används de tidigare stegen för att sedan generera en 3D-modell skapad med 3D Gaussian Splatting. Träningstiden beror främst på antalet bilder samt deras upplösning. Ett större antal bilder i högre upplösning medför en ökad datamängd, vilket resulterar i längre träningstid. Andra faktorer, såsom den använda hårdvarans prestanda och valda parametrar i träningskonfigurationen, påverkar även träningstiden [17].



Figur 3.4: Figuren visar träningen av 3D Gaussian Splatting (3DGS). Den procentuella tiden för det aktuella steget visas längst upp i figuren via en förloppsindikatorn. [17]. Återgiven med tillstånd.

3.1.5 Redigering av 3DGS-modell

Programvaran Postshot har flera inbyggda modifikationsmöjligheter för att anpassa modellen efter behov. Dessa funktioner inkluderar bland annat möjligheten att radera enstaka 3D-gaussians med hjälp av penselverktyget, som markerar de områden som ska tas bort (se figur 3.5) [18].

Figur 3.5: Visualisering av svävande 3D-gaussians som markerats för borttagning med hjälp av penselverktyget i Postshot. [19]. Återgiven med tillstånd.

3.1.6 Exportering av 3DGS-modell

Efter genomförd träning erbjuder Postshot möjligheten att exportera den slutförda 3D-modellen i Polygon File Format (PLY) [20]. För att lagra 3DGS-modeller används filformatet PLY, som lagrar varje enskild gaussian som en punkt i 3D-rymden, med information om position, färg, rotation, skala och opacitet [21].

3.2 Bildtagning för 3D Gaussian Splatting

För att skapa 3DGS-modeller har olika fototekniker använts beroende på om det var ett rum eller ett enskilt objekt som skulle fångas. Säkerställandet av högkvalitativ data för modelleringen krävde även noggrant valda kamerainställningar. Nedan följer inställningarna och metoderna som använts för att fånga data.

3.2.1 Kamerainställningar

3DGS kräver överlappande bilder för att identifiera matchande nyckelpunkter till SfM-algoritmen, se avsnitt 2.2.4. För att garantera att bilderna överlappar tillräckligt mycket användes videoinspelningar, istället för upprepande fotografering, då en video är en uppsättning bilder tagna kort efter varandra.

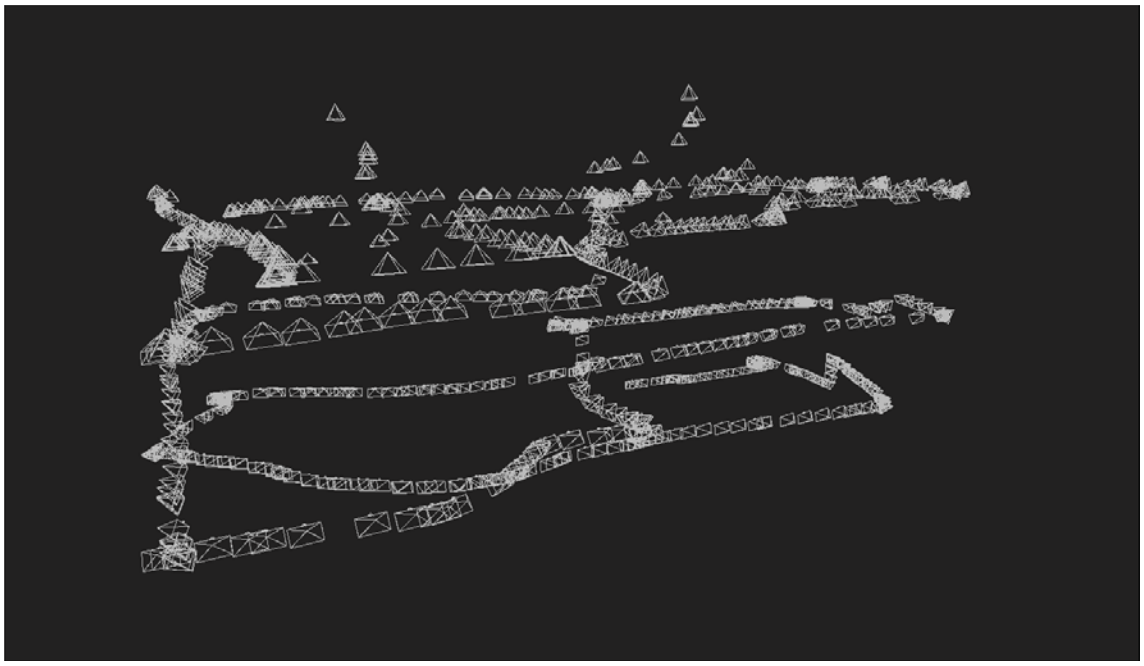
Välexponerade bilder krävs för att en SfM-algoritm ska kunna identifiera detaljer och generera ett exakt punktmoln, se avsnitt 2.2.2. Underexponerade bilder tappar detaljer i mörka områden, medan överexponerade bilder tappar detaljer i ljusa områden. Därför användes manuella kamerainställningar vilket tillåter full kontroll över exponeringen. Följande inställningar justerades och följande riktlinjer användes.

- ISO är kamerasensorns ljuskänslighet och påverkar exponeringen. För hög ISO kan introducera brus och inställningen hölls därför så låg som möjligt och översteg aldrig 400.
- Aperture, eller bländare på svenska, representerar hur brett kameralinsen öppnar sig och påverkar exponering samt skärpedjupet. Aperture mäts i f-tal där exempelvis f/1,4 representerar en öppen lins som släpper in mycket ljus men ger ett kort skärpedjup. Ett högt f-tal som f/16 släpper in lite ljus men ger längre skärpedjup. F-talet bestämdes utifrån ljussituationen i den miljö som fångades och var den inställning som justerades mest för att få bra exponering.
- Slutartid bestämmer under hur lång tid som ljus fångas vid varje bildtagning och påverkar exponering samt rörelseskärpa. En kortare slutartid släpper in mindre ljus men resulterar i mindre rörelseskärpa. Slutartid sattes till en sextiondels sekund för att minska rörelseskärpa och hålla bilderna i fokus.
- Vitbalansen, som bestämmer färgtemperaturen, anpassades efter ljussituationen i den miljö som fångades för att bilderna skulle likna verkligheten.
- 4K-upplösning användes för att fånga många detaljer.
- Film spelades in med 30 bilder per sekund för att säkerställa tillräckligt överlapp mellan bilderna.

3.2.2 Produktionssystemlabbet

För att filma produktionssystemslabbet (PSL), ett laboratorium på Chalmers tekniska högskola, användes en drönare av modellen DJI Phantom 4 Pro. Drönaren flög hela varv runt rummet utmed väggen och kameran riktades inåt mot rummets mitt för att fånga så mycket av rummets egenskaper som möjligt. Denna rörelse upprepades tre gånger i olika höjder, där det lägsta varvet hade en kameravinkel

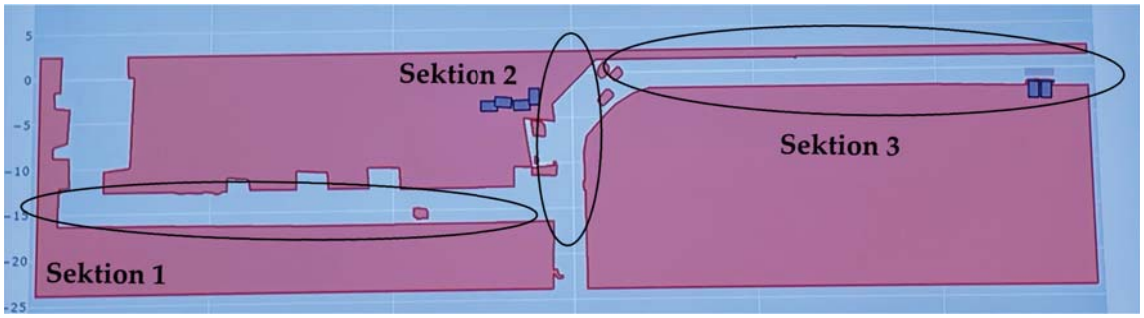
riktad uppåt med 15 grader, mellersta varvet hade en nollgradig kameravinkel och det övre hade en kameravinkel riktad neråt med cirka 30 grader. Sist gjordes en mer fokuserad flygning ovanifrån. Drönaren flög metodiskt över hela golvet genom att flyga rakt fram över hela rummet, för att sedan vända och flyga i motsatt riktning förflyttad i sidled cirka 0.5-1.0 meter. Figur 3.6 visar en illustration av drönarens flygbana. Den totala inspelningstiden uppgick till cirka 17 minuter.



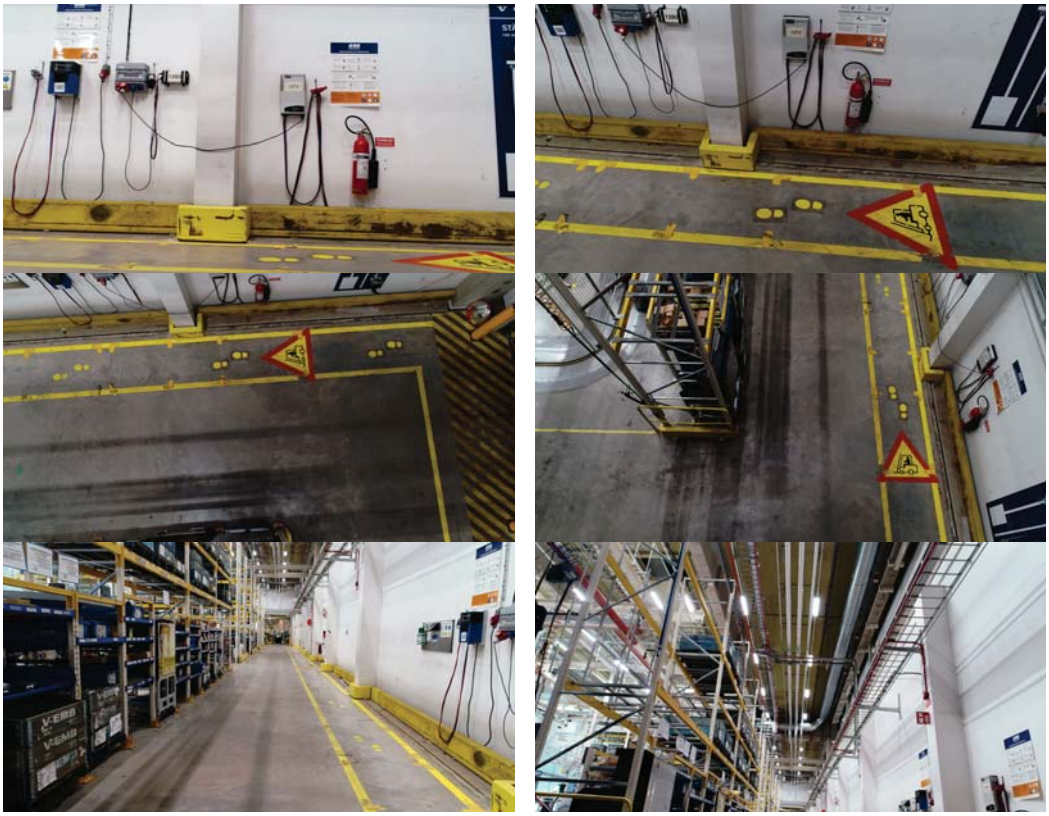
Figur 3.6: Visualisering av kamerans positioner vid drönarflygningen i PSL.

3.2.3 AB Volvos Tuve fabrik

Två DJI Phantom 4 Pro drönare användes för att filma en avgränsad del av AB Volvos fabrik i Tuve, körbanan för autonoma robotar, se appendix A.1. För att underlätta inspelningen delades det aktuella området in i tre sektioner, se figur 3.7. Varje sektion filmades genom att drönaren flög i cirkulära rörelsemönster längs sektionens vägg eller avgränsning på varierande höjder. Kameran riktades rakt fram vid låga höjder och riktades gradvis mer nedåt mot golvet när drönaren steg i höjd för att sist flyga högt upp, rakt ovanför sektionen med kameran riktad nedåt. Den cirkulära flygningen kompletterades med vyer upp mot taket tagna med drönaren i handen. Sist utfördes en genomflygning av endast sektion 1 med drönaren i brösthöjd och kameran riktad rakt fram, vilket upprepades fram och tillbaka genom sektionen. Genomförandet av inspelningen tog totalt cirka fyra timmar. Se figur 3.8 för exempel på bilder fångade av sektion 1.



Figur 3.7: Översiktsbild över området i AB volvos fabrik i Tuve som filmades. Området delades in i tre sektioner, se de ljusblåa inringade delarna som representerar sektion 1, 2 samt 3.



Figur 3.8: Bilder tagna av sektion 1 av AB Volvos fabrik i Tuve, vilket motsvarar den vänstra delen i översiktsbilden i figur 3.7.

3.2.4 Objekt

Objekt filmades både genom att flyga med drönaren samt genom att hålla den i handen och gå runt objekten. För att säkerställa att alla delar av objekten täcktes in genomfördes filmningen i cirkulära rörelser runt varje objekt, ur flera vinklar och på varierande höjdnivåer, vanligtvis fyra till fem gånger beroende på objektets storlek. Vid större objekt krävdes fler rotationer för att fånga in alla detaljer. Detta resulterade i mellan 200 till 400 bilder per objekt, samtliga tagna ur olika vinklar

och höjder. Drönarkamerans position under filmningen av objektet framgår i figur 3.9. Med 3DGS modellerades en gaffeltruck och Volvos autonoma robot, både med och utan last i form av en ljuddämpare.

(a) Exempel på en modellerad robot med en ljuddämpare som last.

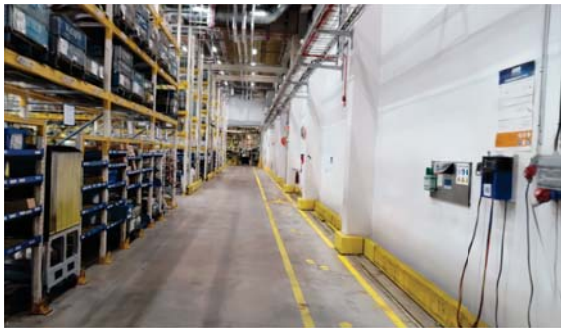
(b) Visualisering av kamerapositionerna runt objektet.

Figur 3.9: Objekt och motsvarande flygbana, som illustreras av kamerapositionerna kring objektet.

3.3 Träning av 3D Gaussian Splatting efter bildtagning

Postshots standardinställningar (se figur 3.1) användes vid träningen av både miljöer och objekt, tillsammans med det rekommenderade maxantalet bilder. Den totala träningstiden för PSL uppgick till cirka två timmar, medan träningen av alla sektioner i AB Volvos fabrik tog cirka fyra timmar. För objekten varierade träningstiden mellan 30 och 60 minuter, beroende på hur många bilder som använts, samt tillgänglig datorkraft. Se appendix A.10 för specifikationerna på datorn som använts. Se figurer 3.10-3.12 för illustrationer av 3DGS-modeller av miljöer och objekt efter avslutad träning.

Figur 3.10: Visualisering av en 3D Gaussian Splatting-modell av PSL efter genomförd träning.



(a) Sektion 1

(b) Sektion 2

(c) Sektion 3

Figur 3.11: 3DGS-modeller av ett begränsat område i AB Volvos fabrik i Tuve uppdelat i tre sektioner.

(a) Gaffeltruck

(b) Robot

Figur 3.12: Visualisering på några 3DGS-modeller av objekt.

Efter träningen användes Postshots inbyggda verktyg för att redigera modellerna. Penselverktyget användes för att ta bort svävande 3D-gaussians samt för att radera omgivningen runt objekten, se figur 3.13 och 3.14.

(a) Markerade **(b)** Raderade

Figur 3.13: Exempel på hur svävande 3D-gaussians identifieras och tas bort från miljöer med hjälp av Postshots inbyggda penselverktyg.

(a) 3DGS modell med omgivning **(b)** 3DGS modell utan omgivning

Figur 3.14: Exempel på hur omgivningen runt ett objekt tas bort från en 3DGS modell med hjälp av penselverktyget i Postshot.

4

Simulering av verkligheten i en digital tvilling

Efter att modellerna av fabriksmiljöerna och objekten skapats, i kapitel 3, beskriver detta kapitel processen för att importera dessa till spelmotorn Unreal Engine 5 (UE5) och bygga upp en dynamisk simuleringsmiljö. Målet är att skapa en virtuell plattform där olika scenarion kan simuleras för att senare generera syntetisk träningsdata. Kapitlet täcker grunderna i UE5, importprocessen av 3DGS-modeller, skapandet och styrningen av digitala karaktärer, samt implementeringen av olika miljömässiga störningar och oförutsedda händelser.

4.1 Simuleringsprogrammet Unreal Engine 5

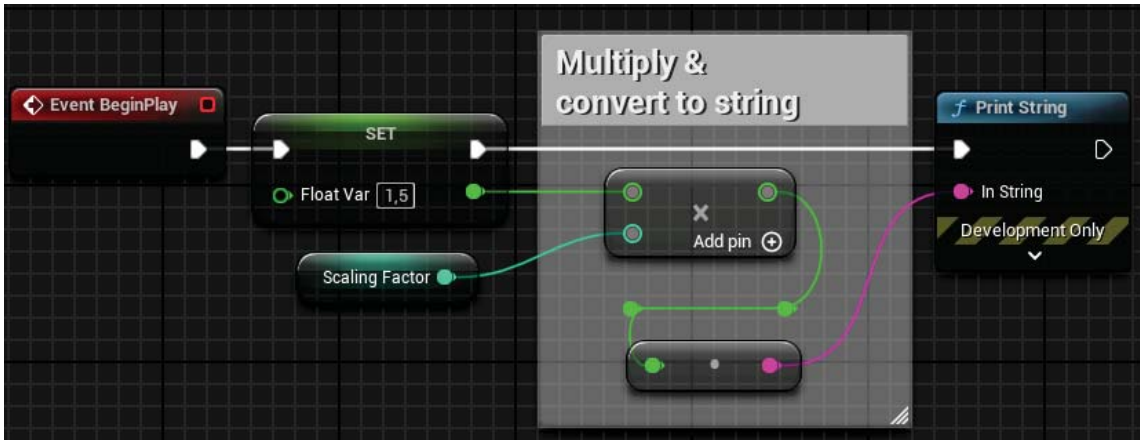
Unreal Engine 5 (UE5) är en spel- och grafikmotor som bygger på det objektorienterade programspråket C++[22]. För att underlätta i utvecklingsprocessen av bland annat spel och simulationer är programmets användargränssnitt baserat på fönster och intuitiva menyer. På så sätt förenklas utvecklingsprocessen då all funktionalitet inte måste skrivas i C++ kod. Användargränssnittet har dock tydliga likheter till C++ kod eftersom båda bygger på objektorienterad programmering. Alla Unrealprojekt kräver ett objekt av klassen World som innehåller allt som ska finnas i spelet. Till World-klassen kopplas ett så kallat gamemode vilket håller koll på spelregler och all data. Ytterligare krävs även ett objekt av klassen Level där en fungerande spelmiljö kan skapas med till exempel karaktärer som interagerar med en omgivning. I en level bygger utvecklaren upp sin värld av en mängd olika objekt som i sin tur tillhör en specifik blueprintklass.

4.1.1 Programmering med blueprints i Unreal Engine 5

Alla typer av objekt som bygger upp en spel- eller simuleringsmiljö i UE5 är av en specifik blueprintklass. Det finns flera färdiga mallar av blueprintklasser som objekt kan ärva. Dessa förenklar utvecklingsprocessen genom att ge tillgång till basal funktionalitet utan att utvecklaren behöver bygga det själv. Den enklaste klassen är actor. En actor används som mall för statiska objekt. Mallen ger all fysik som krävs för att ett statiskt objekt ska kunna interagera med omvärlden. Som påbyggnad på actor finns pawn och character actors, där pawn är en actor som även kan behärskas och styras som en spelkaraktär. På liknande sätt är character en mer avancerad version av en pawn då den har implementerad funktionalitet att kunna

röra sig människolikt i en miljö. Mallarna ger alltså utvecklaren en grund att börja programmera spelet eller simuleringen ifrån.

De flesta blueprints har ett ”graph” fönster där utvecklaren kan programmera objektets funktionalitet. Graphs är ett inbyggt block-programmeringsspråk i UE5 där en mängd fördefinierade C++ funktioner representeras av block som kan kopplas samman för att skapa funktionalitet för det tillhörande objektet. Se figur 4.1 för ett exempel där ett flyttal, FloatVal, sätts till $1,5$ och därefter multipliceras med en känd heltalsvariabel, ScalingFactor. Resultatet skrivs till slut ut i terminalen. Notera att nodernas exekvering sker från vänster till höger enligt den vita exekveringslinjen. Generellt sett sker all programmering av blueprints i graph fönstret men vissa klasspecifika parametrar kan även ändras i objektets menyer som till exempel ”details” fönstret. Se bilaga A.2 för en bild på ett details fönster tillhörande ett kameraobjekt i UE5. Notera att menyn erbjuder ett intuitivt gränssnitt för att, bland annat, justera parametrarna till kamerans storlek (Scale) och synfält (Field Of View).



Figur 4.1: Ett simpelt exempel på blockprogrammering i en graph.

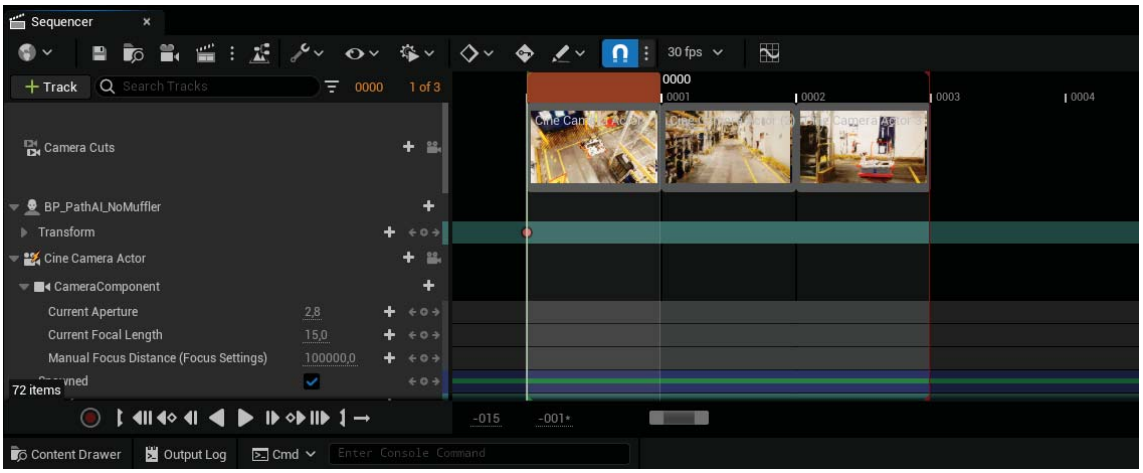
4.1.2 Virtuell fotografering

Förutom att skapa en digital tvilling med hjälp av 3DGS syftade arbetet till att använda den som simuleringsmiljö för att generera ground-truth-data i form av bilder. Den digitala modellen importerades till Unreal Engine 5 (UE5), där virtuella kameror placerades ut och kopplades till en Level Sequence. Därefter renderades stillbilder med hjälp av Movie Render Queue (MRQ), vilket gav noggrann kontroll över renderingsinställningar och effektiv bildexport. En detaljerad beskrivning av arbetsprocessen samt de verktyg som använts presenteras nedan.

Det första steget i arbetsprocessen är att välja kameratyp. I UE5 finns det två huvudsakliga kameratyper: Camera Actor och Cine Camera Actor. Till skillnad från Camera Actor är Cine Camera Actor en mer avancerad variant som efterliknar verkliga filmkameror. Den möjliggör finjustering av en rad parametrar såsom brännvidd, bländare och skärpedjup. Med anledning av dess utökade flexibilitet och omfattande

justeringsmöjligheter valdes Cine Camera Actor som virtuell kamera i simuleringsmiljön.

Nästa steg i processen är att skapa en Level Sequence. Det är ett verktyg i UE5 som används för att strukturera och styra tidsbaserade händelser i spel- eller simuleringsmiljöer. Med hjälp av en Level Sequence kan användaren kontrollera och animera exempelvis karaktärer, objekt och kamerarörelser, samt producera filmsekvenser. Verktyget innehåller en tidslinje som är uppdelad i ett visst antal bildrutor (frames), vilket gör det möjligt att specificera exakt när olika händelser ska inträffa. Exempel på sådana händelser är att en kamera byter vinkel, ett objekt rör sig eller en ljuskälla ändrar intensitet.

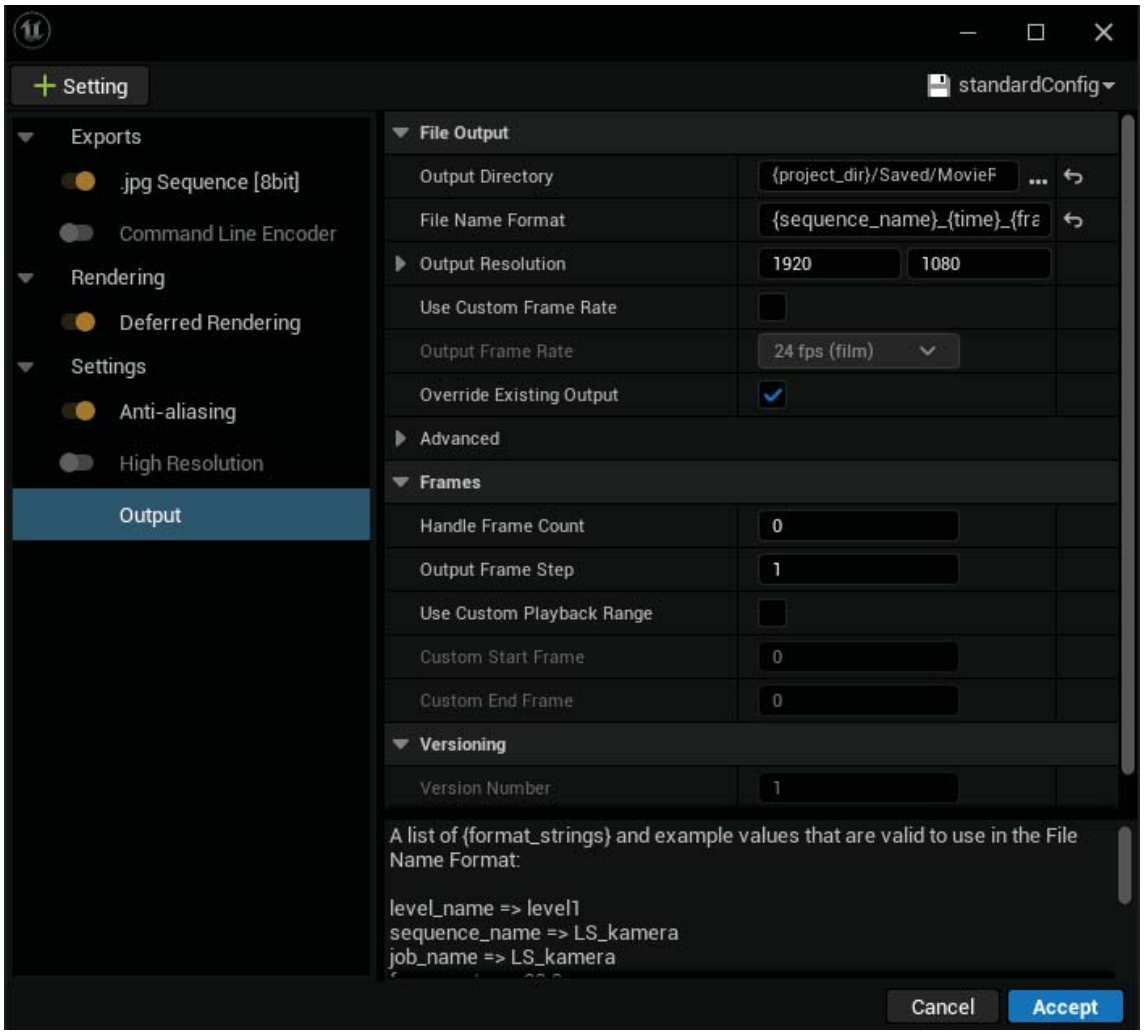


Figur 4.2: Användargränssnittet för Level Sequence i Unreal Engine 5. Verktyget möjliggör styrning av tidsbaserade händelser såsom kamerarörelser, objektanimationer och ljusändringar inom en scen.

En ny Level Sequence skapades via Cinematics-menyn i Content Browser, vilket öppnade arbetsfönstret som visas i figur 4.2, med verktygets tidslinje. Sekvensen var initialt tom, men genom knapparna i verktygsfältet kunde kameror snabbt läggas till. Kamerorna kopplades automatiskt till Level Sequence, och deras positioner i miljön justerades för att fånga de önskade vyerna.

Eftersom arbetet syftade till att rendera stillbilder snarare än filmsekvenser, justerades även arbetsintervallet på tidslinjen så att det endast omfattade ett fåtal bildrutor. Det är möjligt att rendera flera kameror parallellt i UE5, men när detta testades i den digitala tvillingen uppstod problem som gjorde att metoden inte fungerade som avsett. Därför ändrades tillvägagångssättet så att kamerorna renderades sekventiellt, med en bildruta per kamera. Det totala antalet bildrutor i sekvensen motsvarade därmed antalet kameror. Sekvensen renderades sedan genom att använda Render-knappen i verktygsfältet, vilket öppnade Movie Render Queue för vidare arbete.

Movie Render Queue (MRQ) är ett verktyg i UE5 som används för att rendera bilder och videor med hög kvalitet. Det möjliggör detaljerad kontroll av renderingsinställningar samt hantering av flera renderingsuppdrag i en kö. Genom MRQ kan användaren specificera parametrar såsom upplösning, bildfrekvens, filformat och olika postprocess-inställningar. Figur 4.3 skildrar hur verktygets användargränssnitt ser ut vid justering av renderingsinställningar. Ovanstående egenskaper gör MRQ särskilt lämpligt i projekt där detaljerad renderingskontroll och batchhantering efterfrågas.



Figur 4.3: Användargränssnitt för Movie Render Queue (MRQ) i Unreal Engine 5. Verktyget möjliggör detaljerad kontroll över renderingsinställningar såsom upplösning, bildfrekvens, filformat samt postprocess-effekter. Detta är särskilt användbart för högkvalitativa renderingar och batchhantering av flera uppdrag.

Förutom Movie Render Queue finns det flera andra metoder för att ta bilder i UE5. Ett alternativ är funktionen High Resolution Screenshot, som genom en knapptryckning sparar en bild av den aktuella vyn i viewporten. Vid export kan upplösningen

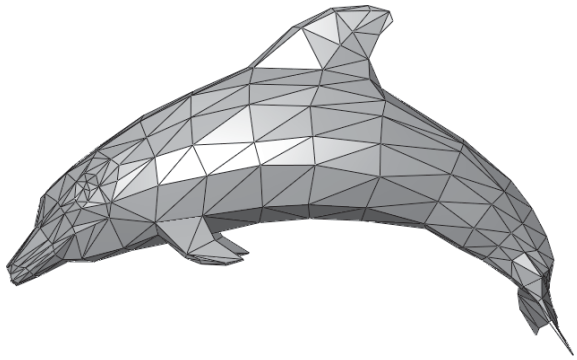
höjas för att förbättra bildkvaliteten. Metoden erbjuder dock begränsade inställningsmöjligheter och lämpar sig främst för enstaka bilder snarare än större bildserier. Ett annat alternativ är att ta en vanlig viewport-screenshot, vilket sparar en bild av skärmens aktuella upplösning utan möjlighet till vidare konfiguration. Ytterligare en metod är att använda path tracing som renderingsmetod. Det ger mer realistiska ljus- och skuggeffekter, men kräver avsevärt mer prestanda och passar därför bäst för mindre projekt eller enskilda bilder.

I detta arbete valdes MRQ eftersom verktyget erbjuder detaljerade inställningsmöjligheter och smidig integration med Blueprints för automatisering, något som de andra metoderna saknar. MRQ är ett inbyggt tillägg i UE5 som aktiveras via Edit-menyn. Efter aktivering kan användaren skapa renderingsjobb, vilka omfattar rendering av en eller flera Level Sequences. Varje jobb tilldelas ett unikt job name, vilket sedan kan användas för att namnge bilderna och organisera dem vid export. Flera renderingsjobb kan läggas till i Render Queue; de köas upp för att köras i sekvens, en process som kallas batch-rendering.

I MRQ kan varje renderingsjobb ha sina egna renderingsinställningar. Inställningar som upplösning, bildfrekvens och filformat kan justeras för att optimera renderingarnas kvalitet och prestanda. För ytterligare bildkvalitet kan postprocess-effekter som motion blur och depth of field appliceras, medan anti-aliasing kan minska trappstegsformade kanter för att förbättra den visuella intrycket. Om högsta möjliga realism önskas kan även path tracing aktiveras. Användaren kan välja filformat för export, såsom .PNG eller .EXR, beroende på projektets behov. Dessutom går det att namnge både bilderna och mappen där de sparas på datorn. I namnen kan olika inbyggda parametrar, såsom job name, sequence name, datum och tid läggas till för att underlätta organiseringen.

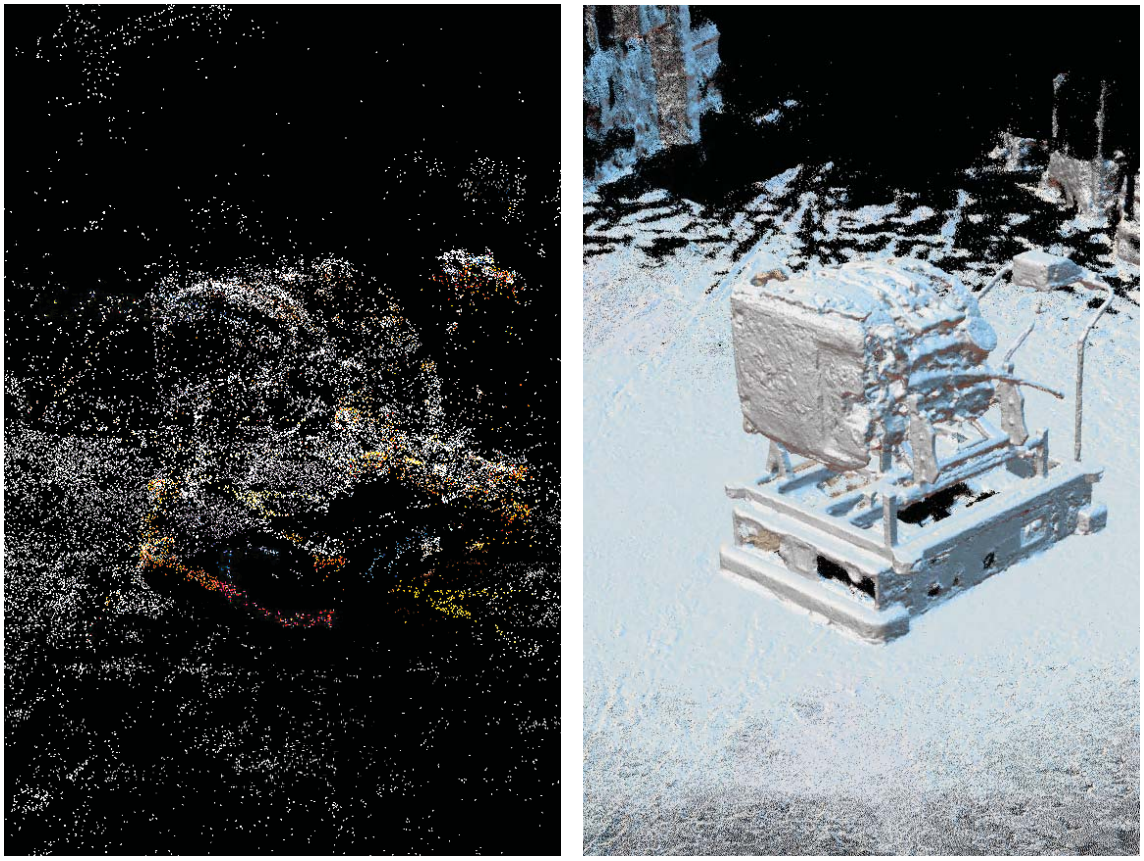
4.1.3 Skapandet av meshar för Unreal Engine 5

Inom UE5 är varje objekts geometri representerad med en mesh. En 3D-mesh, eller 3D-polygonnät, är en digital ytmodell som används för att representera objekt i tredimensionell form. Modellen byggs upp av hörnpunkter, kanter och polygoner. Hörnpunkterna fungerar som koordinater och kanterna kopplar samman närliggande hörnpunkter. Polygonerna, som oftast består av trianglar eller fyrhörningar, omsluter kanterna och skapar därmed objektets yta. Genom att kombinera koordinater, kanter och polygoner bildas en fullständig 3D-mesh [23], se figur 4.4.



Figur 4.4: Exempel på en mesh-modell där hörnpunkter, kanter och polygoner (trianglar i detta fall) skapar objektets yta.

För användning av 3DGS-objekt i UE5 behövs tillhörande mesh-objekt, se avsnitt 4.2.2, vilka har skapats med programmet RealityCapture. Från laserskanning eller en uppsättning överlappande bilder skapar RealityCapture ett punktmoln utifrån de tekniker som introducerades i avsnitten 2.2.1 och 2.2.2. Programmet har möjlighet att utifrån punktmolnet skapa en mesh-modell, se figur. 4.5. Inbyggda verktyg finns för att förbättra modellen och ta bort överflödig geometri.



(a) Punktmoln **(b)** Mesh

Figur 4.5: Exempel på hur RealityCapture går från punktmoln till mesh.

4.2 Importering av 3DGS-modell till Unreal Engine 5

I detta avsnitt presenteras de verktyg och metoder som behövs för att importera 3DGS-modeller till Unreal Engine 5.

4.2.1 Tredjepartstillägg

UE5 innehåller flera verktyg och tekniker för att bygga upp en simulerad värld. Utöver dessa verktyg finns det tredjepartstillägg, även kallat plugins, som utökar dess funktionalitet. För att importera 3DGS-modeller i UE5 krävs ett tillägg, några exempel är Luma AI, XVERSE och 3D Gaussians Plugin [24]. Dessa tillägg läser in 3DGS-filer av typen PLY (polygon file format) och omvandlar det till ett grafiskt objekt i UE5.

Tilläggen är mycket lika men har även vissa skillnader. Den största skillnaden mellan dem är att Luma AI omvandlar samma PLY-fil till fyra olika objekt med unika egenskaper. En av dessa egenskaper erhålls hos det dynamiska objektet som tar hänsyn till ljussättningen, i den uppbyggda miljön, i UE5. Detta är en fördelaktig egenskap då det möjliggör för en bättre representation av verkligheten. Denna egenskap saknas i de andra tilläggen där färgerna i 3DGS-modellen är av en förutbestämd fast styrka. Därmed används tillägget Luma AI i detta projekt.

4.2.2 Justering & uppbyggnad av en modellmiljö

Då modellen av en miljö importerats i UE5 ska den justeras och anpassas för att möjliggöra de kommande delarna i implementationen. Eftersom 3DGS-modeller saknar korrekt storlek och oftast inte är centrerade gäller det att genomföra följande steg. Det första steget är att rotera och translatera modellen så att marken ligger i samma plan som xy-planet i UE5.

I det andra steget gäller det att förbereda modellen för att hantera digitala karaktärer och segmenteringsalgoritmen som skapas i kapitel 5. Eftersom 3DGS inte har någon kollision behöver karakträrmodellerna representeras av solida mesh-objekt för att kunna interagera med omgivningen. Därmed behöver även golven och väggarna representeras av solida objekt. Därför skapas en actor blueprint med en static mesh-komponent i form av ett rätblock. För att bygga ett golv som karaktärerna kan stå på placeras rätblocket i xy-planet och förstoras längs med x- och y-axeln för att breda ut ett plan över golvet. Modellens väggar byggs upp på liknande sätt men roteras och translateras enligt behov för att täcka en vägg.

Alla rätblock som placerats i världen har önskad kollisionsdata men även en visuell representation. Eftersom de utplacerade väggarna samt golvet blockerar 3DGS-modellen, se figur 4.6 (a), behöver de göras osynliga under simulationens gång. Dess synlighet växlas till osynlig då simulationen startar, se Figur 4.6 (b).

(a) Modellens uppbyggnad med synliga väggar och golv representerade av rätblock.

(b) En bild där rätblocken som representerar väggar och golv gjorts osynliga under simulationens gång.

Figur 4.6: En visuell jämförelse mellan hur den digitala modellen är uppbyggd och hur den representeras visuellt under simulationens gång.

4.3 Digitala karaktärer

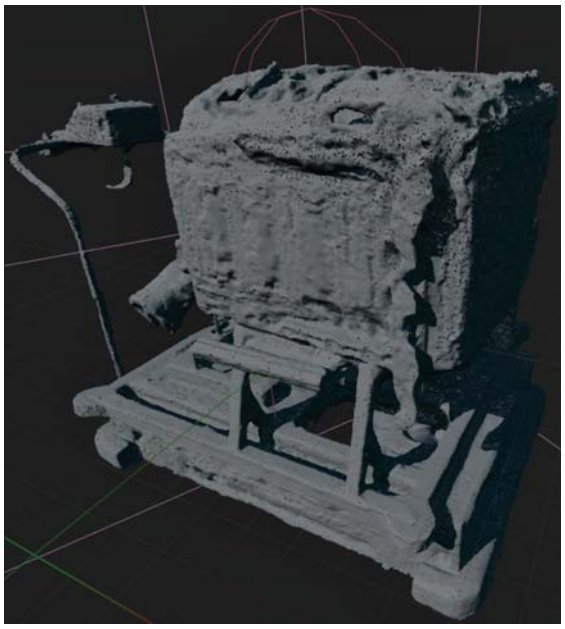
Att återskapa verkligheten i en digital miljö kräver möjligheten att simulera dynamiska objekt. Dessa dynamiska objekt kan bland annat representeras som fordon eller människor. Nedan presenteras ett tillvägagångssätt för att kombinera digitala karaktärer och 3DGS-modeller i spelmotorn UE5.

4.3.1 Grafik och kollision

För att skapa en karaktär tilldelas ett objekt, av typen character blueprint, grafik och kollision. Character-objektet bygger upp karaktärens generella struktur med fokus på den visuella delen. Ytterligare funktionalitet hos karaktärer skapas i andra objekt som länkas till ett character-objekt eller en specifik instans. Ett exempel är klassen AIController som länkar samman karaktärens beteende med den visuella representationen hos character objektet.

För att tilldela en 3DGS modell och kollision till en character blueprint importeras en modell och tillhörande mesh till UE5, se Figur 4.7 (a) respektive (b). 3DGS modellen tilldelas till karaktären genom att dra och släppa objektet i component fältet. Därefter skalas modellen till sin verkliga storlek, eftersom Postshot endast estimerar storleken vid modellens skapande. Karaktärens kollisionsdata implementeras genom att lägga till en static mesh i components och tilldela det önskade objektets mesh till static mesh komponenten. Därefter skalas, roteras och translateras komponenten för att, i största möjliga mån, överensstämma med grafikmodellens storlek, riktning respektive position i rummet. Tillslut framställs en karaktär med grafik och mesh enligt figur 4.7 (c).

(a) 3DGS modell i UE5



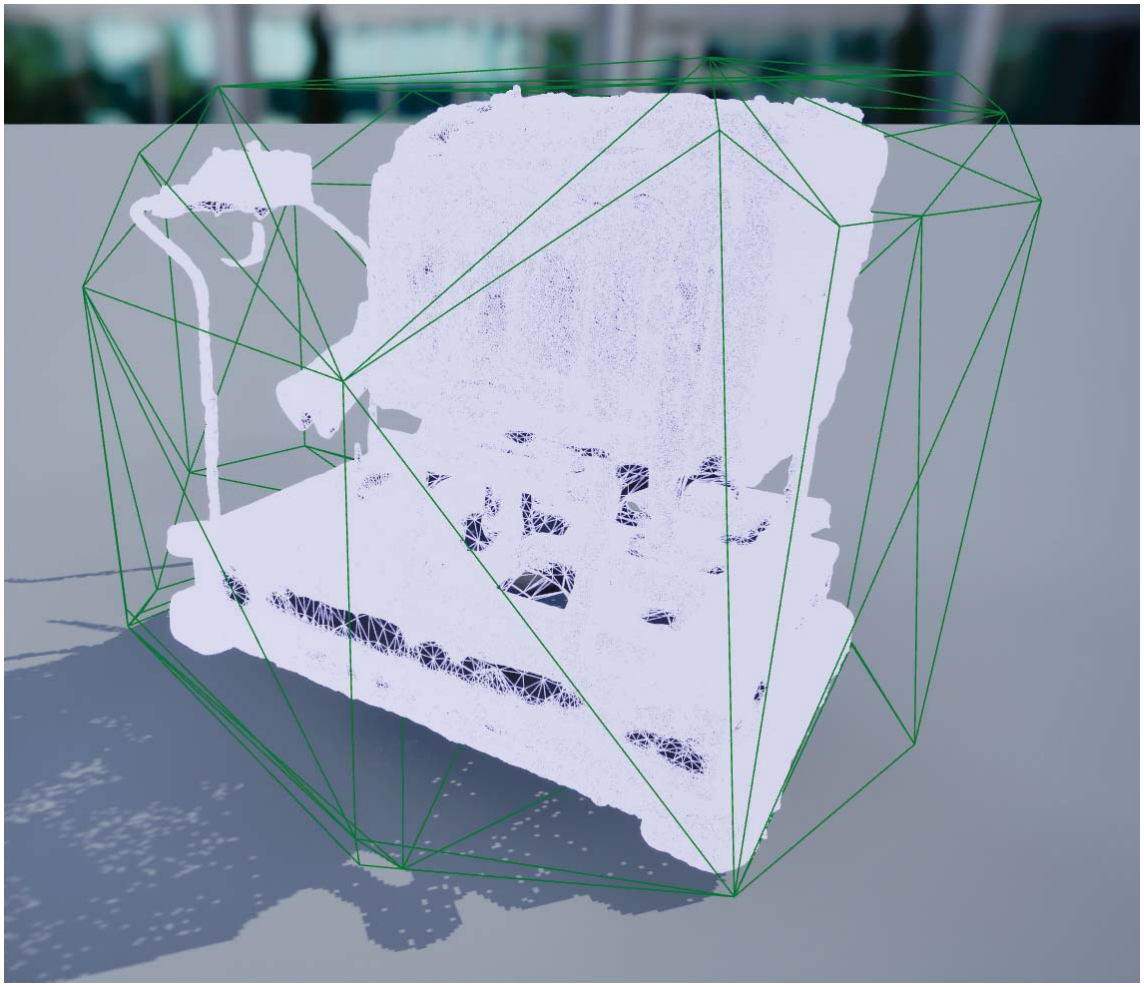
(b) Objektets mesh i UE5

(c) Sammanställd karaktär i UE5

Figur 4.7: Grafik för en 3DGS-modell (a) och tillhörande mesh (b) av ett objekt som sammanställts till en fullständig karaktär i (c).

Eftersom en mesh är synlig som standard är det även viktigt att göra komponenten osynlig under simulationens gång. Detta säkerställs genom att i character blueprint växla static mesh komponentens synlighet vid simulationens start med hjälp av funktionen SetVisibility.

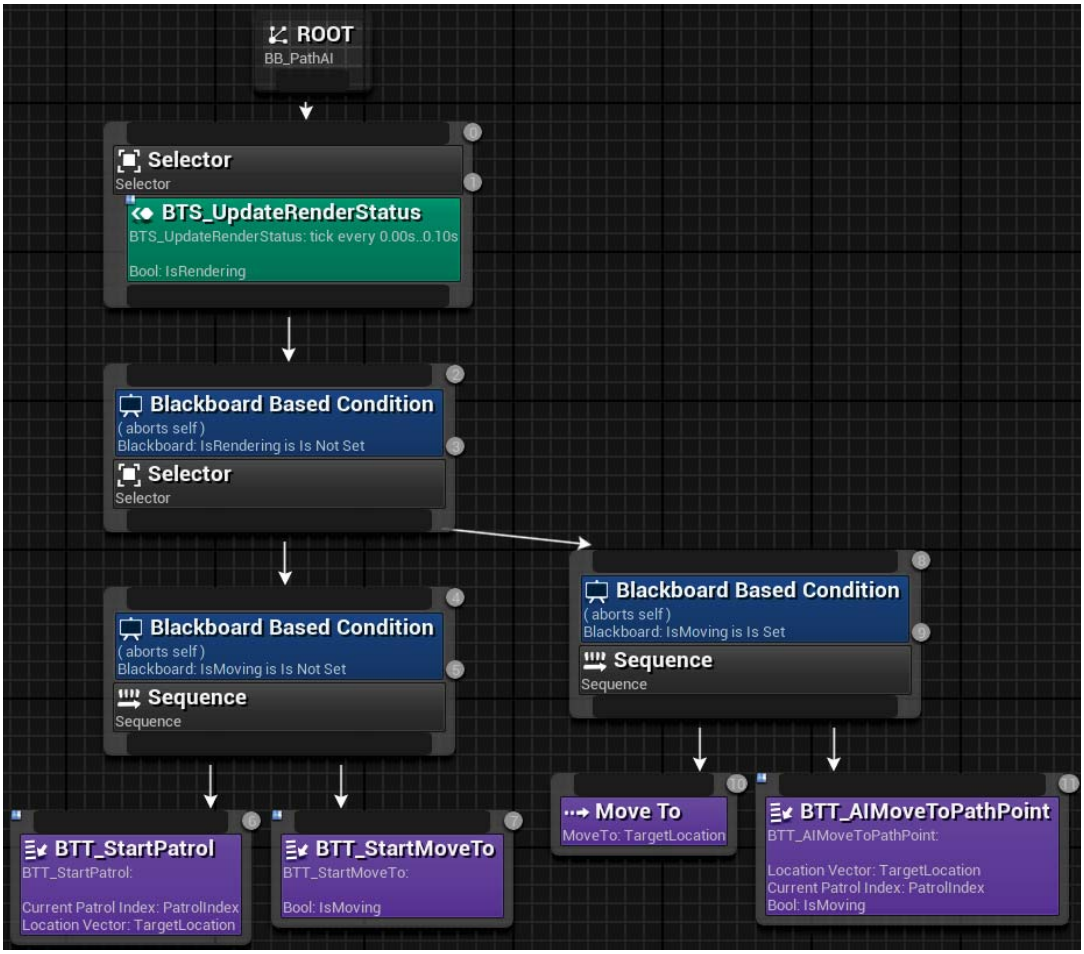
För att optimera simulationen antar UE5 vissa förenklingar med målet att minska beräkningskraft. En sådan förenkling är att importerad static mesh-kollision förenklas till färre polygonytor runtom objektet. Detta innebär att objektet tar upp mer plats i simulationen än vad som visas av grafiken, se de stora polygonytorna mellan de gröna linjerna runtom roboten i figur 4.8. För att erhålla en så verklighetstrogen representation som möjligt bör därför den komplexa kollisionen användas. Detta säkerställer att objektet inte tar upp mer plats i simulationen än sin verkliga motsvarighet. Det vill säga att alla polygonytor som bygger upp objektets mesh används för att definiera kollisionen, se de vita linjerna på roboten i figur 4.8. Denna definitionsändring görs i mesh blueprint, i details panelen under collision där collision complexity väljs till ”Use Complex Collision As Simple”.



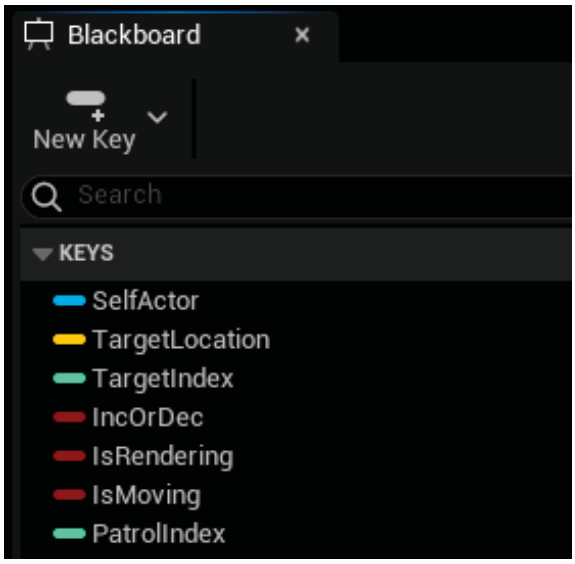
Figur 4.8: En visuell representation av polygonstrukturen för simpel (grön) och komplex (vit) kollision hos en robot i Unreal Engine 5.

4.3.2 Karaktärers beteende

Karaktärer kontrolleras med hjälp av en styrenhet, AIController, som kopplar samman en karaktär med en specifik beteendeenhet, behaviour. Dess behaviour är i sin tur kopplad till ett behaviour tree och en så kallad blackboard, se Figur 4.9. Dessa två är tätt sammankopplade och utgör den största delen av hur karaktären agerar under simuleringen. Behaviour tree byggs upp av olika sekvenser som är villkorsbaserade. Dessa villkor baseras på variabler, så kallade ”keys”, som kontinuerligt uppdateras i blackboard beroende på sekvensernas utförande eller utomstående faktorer. En av dessa variabler används för att säkerställa att karaktären endast väljer ut ett nytt mål efter att den nått sitt nuvarande. Därför växlas en boolean key ”IsMoving”, i blackboard, på och av varje gång ett nytt mål hämtas respektive nås av karaktären. På samma sätt sparas bland annat vektorn ”TargetLocation” som innehar x, y och z koordinater till målet i blackboard, se variabeldeklarationerna i Figur 4.10. Därmed kan målet hämtas av noderna i behaviour tree för att planera en rutt samt utvärdera om karaktären nått det.



Figur 4.9: Behaviour tree för en implementation av en digital karaktär. Till vänster: initiera ett nytt mål. Till höger: flytta karaktären till dess mål



Figur 4.10: Deklarationer av keys i en Blackboard där färgerna indikerar dess typ. Blå: object, Gul: vektor, Grön: int, Röd: bool.

I implementeringen av digitala karaktärer skapas två typer av beteenden. Ett beteende där karaktären slumpmässigt väljer ut punkter att färdas till, inom en specifik radie, och ett där den följer en bana av utplacerade punkter i världen. Vardera behaviour tree består av två sekvenser. En sekvens som väljer ett nytt mål i världen och en sekvens som flyttar karaktären till det målet. För att karaktären ska kunna planera en rutt från dess nuvarande position till målet behöver världens gränser bestämmas. Detta hanterar NavMeshBounds Volume vilket är ett objekt i UE5 som definierar en yta längs marken där karaktärer kan röra sig. Volymen placeras ut på de öppna ytorna i en digital modell där karaktärerna får befinna sig. För att täcka alla öppna ytor utan att överlappa med hinder används flera volymer för att definiera ett heltäckande område. Golvet och väggarna i den uppbyggda miljön som beskrivs i avsnitt 4.2.2 används för att ge NavMeshBounds Volume möjligheten att bestämma en yta över golvet, se Figur 4.11.

(a) NavMesh med väggar **(b)** NavMesh i en 3DGS miljö

Figur 4.11: Utplacerad NavMeshBounds Volume där den gröna ytan visar en navigerbar yta för en digital karaktär. (a) visar hur ytan förhåller sig till golv och väggar i volymen. (b) visar en navigerbar yta i en 3DGS miljö.

4.4 Simulering av störningar och ovanliga situationer

För att uppnå en verklighetstrogen simuleringsmiljö är det väsentligt att återskapa de förhållanden och detaljer som karaktäriserar den fysiska miljön. En metod för detta är att skapa en digital tvilling och sedan simulera dynamiska förhållanden och specifika detaljer. I en fabriksmiljö kan detta till exempel omfatta objekt som flyttas, skiftande ljusförhållanden, damm och spindelnät.

Utöver återskapande av detaljer och tidsvarierande inslag kan miljön användas för att simulera situationer som kan uppstå. Detta kan sedermera nyttjas för testning och optimering av olika system, såsom produktionsflöden och säkerhetsrutiner [25]. I nästkommande avsnitt beskrivs hur dynamiska förhållanden, störningar och olika scenarier har återskapats i simuleringsmiljön i Unreal Engine 5 (UE5).

4.4.1 Skiftande ljusförhållanden

I en fabriksmiljö varierar ljusförhållandena beroende på tid på dygnet, väder, typ av belysning och dess placering. Ljussättning spelar en central roll för hur en omgivning upplevs och hur objekt och detaljer framträder. Genom att simulera de ljusvariationer som kan förekomma kan en mer verklighetstrogen digital tvilling skapas.

UE5 erbjuder flera sätt att simulera ljus, exempelvis genom inbyggda ljuskällor som Point Light, Spot Light, Sky Light och Directional Light. Skillnader i ljuskällornas egenskaper gör att de påverkar miljön på olika sätt och lämpar sig därför för olika ändamål. Directional Lights sänder parallella ljusstrålar från ett oändligt avstånd och sprider ett jämnt ljus över hela scenen [26]. Deras främsta syfte är att återskapa solljus, vilket gör dem passande för simulering av ljusvariationer.

Många av standardmallarna för projekt i UE5 innehåller en mapp som heter Lighting, där det finns en Directional Light som representerar solen. När ljuskällan markeras öppnas dess Details-panel som visar olika inställningar för det aktuella objektet. Genom att justera värdena på Intensity och Temperature simulerades skiftande ljusförhållanden i form av ändrad ljusintensitet och färgtemperatur.

4.4.2 Damm, spindelnät och andra störningar

Vid simuleringen av damm, spindelnät och andra visuella störningar som kan förekomma i en fabriksmiljö användes Dirt Masks. En Dirt Mask är en textur som består av exempelvis damm eller smuts och har en transparent bakgrund [27]. Genom att applicera texturen på de virtuella kamerornas linser i UE5, skapas en effekt av en lins med störningar, vilket påverkar bildernas utseende. Det finns flera inbyggda Dirt Mask-texturer att välja mellan, men det går även att skapa egna eller ladda ner färdiga. De färdigskapta texturerna som användes i arbetet laddades ner från [28]. En sådan textur kan därefter användas på enskilda eller samtliga kameror i

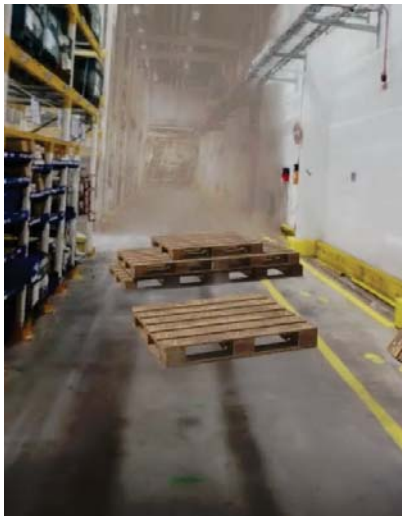
scenen. Dessutom kan olika kameror ha olika Dirt Masks och varje texturs intensitet och färg kan justeras för att uppnå önskad visuell effekt.

En Dirt Mask appliceras på en kamera genom att aktivera Dirt Mask Texture i kamerainställningarna. Därefter kan önskad textur väljas med hjälp av rullgardinsmenyn. För att skapa en egen Dirt Mask behövs en bild i PNG-format med transparent bakgrund. Vid import till UE5 konverteras den automatiskt till en tvådimensionell bildtextur (Texture2D), och blir därefter tillgänglig för val i Dirt Mask-menyn.

4.4.3 Simulering av hinder och oförutsedda händelser

För att representera de utmaningar autonoma robotar kan möta i fabriksmiljöer, implementerades scenarion i UE5 där framkomligheten begränsas. Ett exempel innefattar blockering av robotars färdväg med objekt som pallar, se figur 4.12. Pallen, som är en 3D-modell, hämtades från Unreal Engine Marketplace (FAB) och importerades till projektet. Dessa simulerade hinder placerades därefter i miljön.

Potentialen hos den digitala tvillingen för säkerhetsrelaterade applikationer undersöktes genom att simulera en brand i UE5, se figur 4.12. För att skapa realistiska brandscenarier användes en inbyggd brandfunktion från standardbiblioteket ”Starter Content” i UE5. Vid simuleringen varierades parametrar som eldens storlek, intensitet och rökutveckling för att skapa olika bränder. Kamerapositioner varierades även för att fånga branden från olika observationsvinklar, vilket möjliggör generering av visuell data av brandförlopp.



(a) Visuell representation av simulerad pall och rök.

(b) Visuell representation av simulerad eld.

Figur 4.12: Simulering av pallar och en brand i UE5.

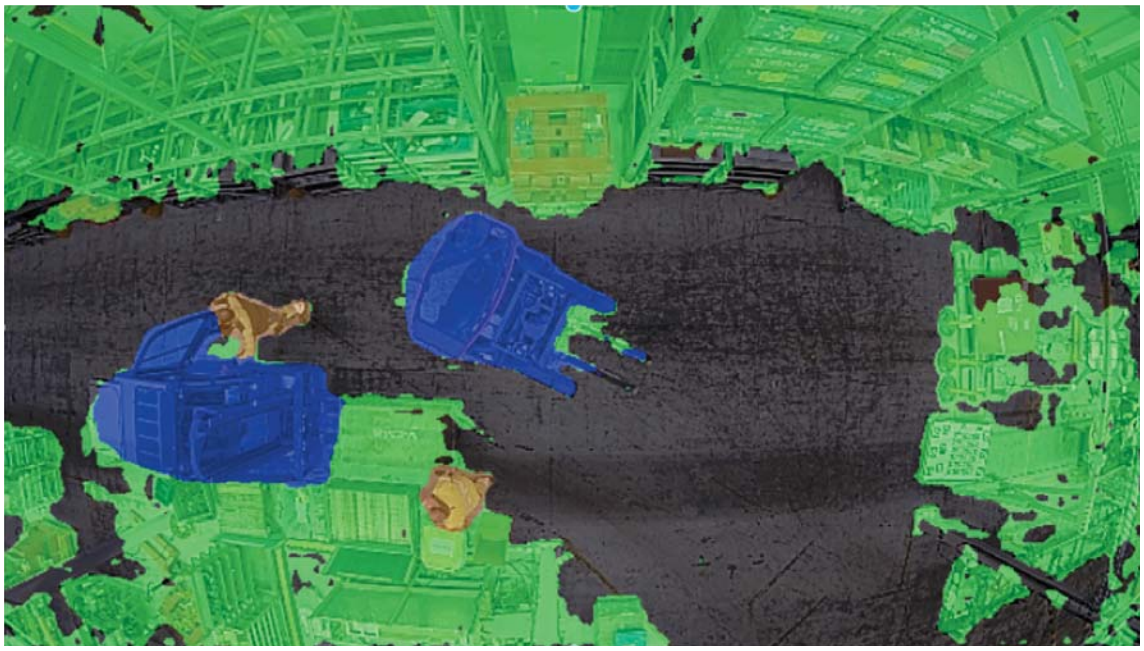
5

Bildgenerering och segmentering av data

Genom att använda 3D Gaussian Splatting (3DGS) kan man simulera det grafiska och få simulering närmare verkligheten. Detta gör det inte bara lättare för människor att analysera, utan ger också möjligheten att skapa simulerade bilder som stämmer väl överens med verkligheten. Dessa bilder kan användas för att träna AI-modeller som använder sig av bilder för att göra till exempel semantisk segmentering eller objekt-igenkänning. I detta kapitel beskrivs implementationen för att generera ground-truth, från simuleringen av Volvos fabrik och robotar, för en semantisk segmenteringsmodell. Därefter skapas en semantisk segmenteringsmodell som tränas på den genererade datan.

5.1 Semantisk segmentering

Semantisk segmentering är en metod för att lära datorer att visuellt klassificera objekt. Tekniken använder en deep learning-algoritm för att identifiera och klassificera objekt; därefter tilldelar den en etikett till varje pixel i bilden, se figur 5.1. Semantiska segmenteringsmodeller tar in en bild och genom ett komplext neuralt nätverk skapar den en segmenteringskarta, där varje pixel i bilden kodats baserat på de semantiska klasser som identifierats [29]. Googles Deeplab-modell är ett exempel på en sådan modell som visat hög precision [30].



Figur 5.1: Semantisk segmentering från aktuell situation på Volvo Trucks i Tuve, Där hinder(grön), truckar(blå) och människor(röd) segmenteras ut.

För träning av semantiska segmenteringsalgoritmer krävs stora mängder annoterad data. Inom datavetenskap kallas data av detta slag för ground-truth. Denna typ av data fungerar som ett facit under träning och låter maskininlärningsmodellen jämföra sin prediktion med vad som är korrekt. Det är därför viktigt att ground-truth-data baseras på observationer från verkligheten, samt att insamlingen och annoteringen av data utförs noggrant [31].

Fördelen med semantisk segmentering gentemot andra objektdetekterings-tekniker är dess precision av objektens form och storlek. Detta gör den lämplig i situationer där en mer precis position krävs, som till exempel i autonoma fordon. Nackdelarna är att det är mer beräkningstungt och kräver ground-truth där varje pixel är klassad. Detta gör skapandet av ground-truth betydligt svårare och därför används ofta simulerad data och Unsupervised Domain Adaptation[32].

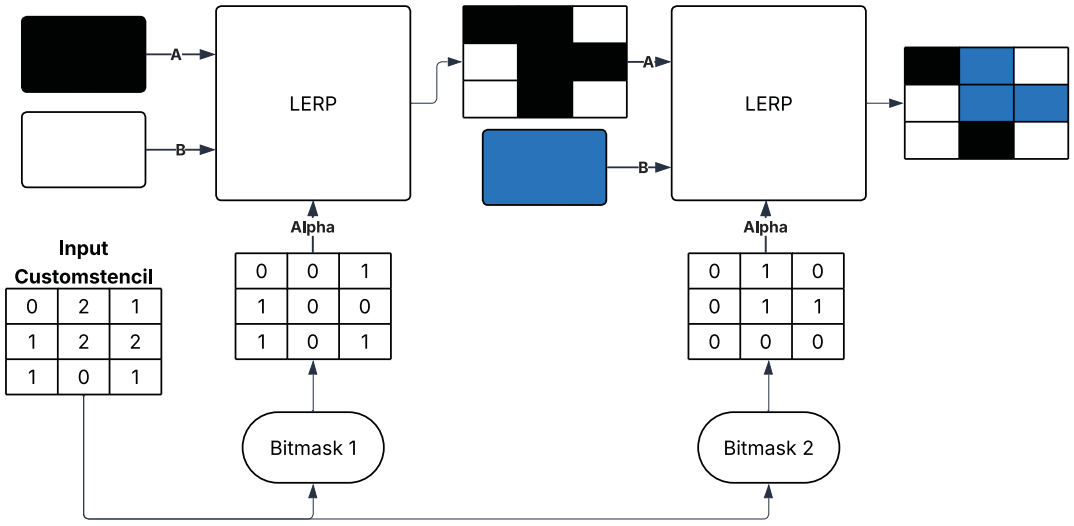
5.2 Generering av semantisk segmenteringsdata från UE5 simulering

Nedan presenteras hur semantisk segmenteringsdata genereras från simuleringsmiljön som byggdes upp i kapitel 4. Med hjälp av postprocessing-material skapas färgkodade bilder baserat på objektklasser som sedan omvandlas till segmenteringskartor genom ett pythonscript.

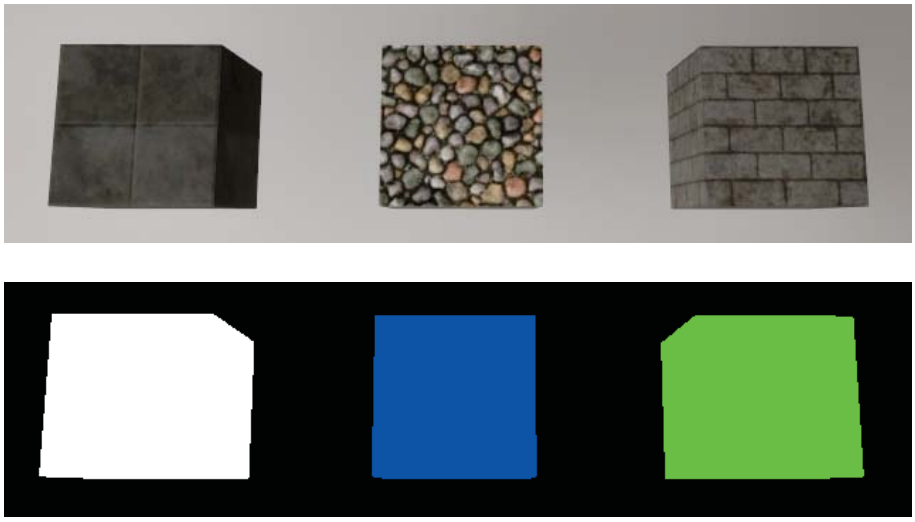
5.2.1 Postprocessing och postprocessing-material

Unreal Engine 5 använder en grafik-pipeline, bestående av flera steg, som hanterar olika delar av grafiken, såsom djup, transparans och färg för att sammanställa bilden som visas på skärmen. Det sista steget i pipelinen är postprocessing vilket används för att manuellt lägga till egna och slutgiltiga effekter på grafiken som till exempel glöd eller glans. Resultatet från postprocessing-steget är den slutgiltiga bilden som kommer visas. För att påverka postprocessing-steget skapas ett postprocessing-material. Materialet har endast en output, Emissive color, vilket bestämmer hur objektet ska glöda. Postprocessing-materialet läggs till i kameran och agerar som en lins för renderingen. Postprocessing-materialet appliceras på alla texturer i grafiken men beroende på hur man bygger upp sitt material påverkas enskilda objekt istället. Genom att skapa ett postprocessing-material som blir olika färg beroende på vilken klass objektet har kan man skapa en segmenterad bild.

Materialet i bilaga A.6 kan segmentera ut objekt till fyra olika klasser beroende på vilka värden de har på parametern custom stencil. En custom stencil är ett heltal mellan 0 och 255 som används för att markera objekt så att de kan särskiljas vid renderingen. Genom att använda SceneTexture:Custom Stencil erhålls en matris som innehåller vilken custom stencil som befinner sig på varje pixel. Denna matris skalas och kopplas till de olika bitmaskerna som gör matrisen till endast ettor och nollor beroende på om det matchar det andra värdet som matas in i bitmasken. Matriserna används därefter som alpha för Lerp-noderna. Lerp noden fungerar genom att den slår samman två olika grafiker A och B beroende av alpha. Om alpha är talet ett så kommer bara B att returneras medan om alpha är talet noll returneras A, detta sker för varje pixel, se figur 5.2. Inledningsvis gör materialet hela grafiken svart. Därefter appliceras Lerp på den svarta grafiken med första klassfärgen, där den första bitmasken används som alpha. I nästa steg appliceras Lerp på det tidigare resultatet, fast med den andra klassificerings-färgen. Då användes den andra bitmasken som alpha. Detta görs för alla klasser och slutligen kopplas resultatet till Emissive color och man får ett resultat som i figur 5.3. Detta postprocessing-material används i följande avsnitt för att segmentera data från UE5.



Figur 5.2: Figuren visar hur postprocessing-materialet fungerar. Först appliceras Lerp på svart och vit färg, där alfa-kanalen utgörs av en matris som innehåller värdet 1 på de positioner där custom stencil värdet är 1, och 0 på övriga positioner. Därefter appliceras lerp på resultatet från den tidigare lerp-noden och färgen blå med en matris är alfa-kanalen utgörs av en matris som innehåller värdet 1 på de positioner där custom stencil värdet är 2, och 0 på övriga positioner. Till slut retuneras grafik där pixlarna har färgkodats beroende på custom stencil värdet

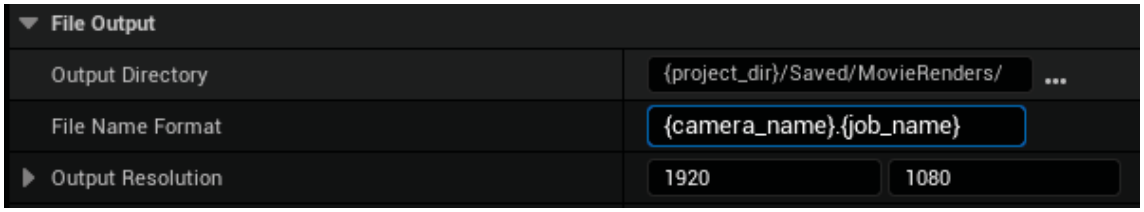


Figur 5.3: Exempel på hur postprocessing-materialet segmenterar tre olika kuber till tre olika klasser. Övre bilden visar kuberna innan segmentering och undre bilden visar kuberna efter segmenteringen.

5.2.2 Simuleringsmiljö och viktiga funktioner

Simuleringsmiljön som används är den som skapats i kapitel 4. Både för att ta och exportera bilder ifrån simuleringen används den inbyggda UE5 kameran Cineactor tillsammans med en Level Sequence där en bildruta läggs in för varje kamera. Kamerorna placeras manuellt ut inuti simuleringsmiljön så att de får önskade vyer. Därefter importeras ett valfritt antal digitala karaktärer med det slumpmässiga rörelsebeteendet som beskrivs i kapitel 4.3.2.

För att automatisera renderingen av bilder används tillägget Movie Render Queue, som beskrivs i avsnitt 4.1.2. Med hjälp av Movie Render Queue och blueprints kan man skapa renderingsjobb som annars behöver göras manuellt. För att skapa och konfigurera ett renderingsjobb har en funktion skapats, se bilaga A.7. I ”allocate job” väljs den Level Sequence som skapats tidigare, vilket genererar ett renderingsjobb vars namn automatiskt baseras på bildens sekventiella nummer. Därefter sätts konfigurationen, som i figur 5.4. Slutligen renderas bilden och sparas i den mapp som specificerats i konfigurationen.



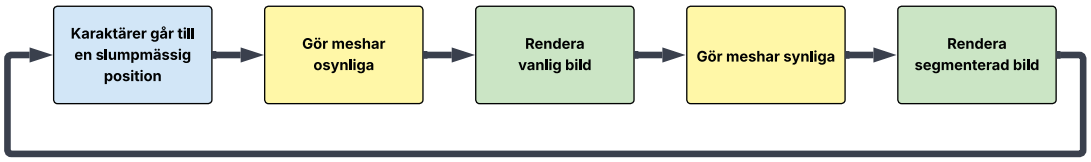
Figur 5.4: I figuren synns konfigurationen som används vid bildrenderingen. Filnamnet sätts till en kombination av kamerans namn och renderings jobbets namn som anger bildens ordningsnummer. Ett exempel på ett filnamn som genereras är $\text{Camera1}_1.png$

För att skapa den segmenterade bilden används samma funktion fast med en annan konfiguration. I konfigurationen läggs postprocessing-materialet till och man markerar att man vill använda 32 bitars postprocessings material för att behålla kvalitén. Segmenteringen kommer inte ske av 3DGS-modellen eftersom den är full med genomskinliga gaussians, som hade skapat en felaktig segmentering. Därför kommer istället mesharna, som används för kollisionen i kaptiel 4, att segmenteras beroende av klass. Men eftersom dessa meshar har gjorts till osynliga så kommer de att optimeras bort från renderingspipelinen och inte påverkas av postprocessing-steget. Därför skapas en funktion som kan växla mesharna mellan synliga och osynliga, som kan kallas mellan renderingen av den riktiga bilden och den segmenterade, se bilaga A.8. Funktionen hämtar alla karaktärer i simuleringen och loopar därefter igenom dem och sätter deras meshar till synliga eller osynliga beroende på funktionens inparametern visibility. Sedan sker detsamma för alla väggar och deras meshar.

För att karaktärerna ska befinna sig på samma ställe vid renderingen av båda bilderna skapas ett villkor som gör att de står stilla när de har nåt sin position. Ett event kopplas till renderingen, som ändrar villkoret när bilden har renderat klart.

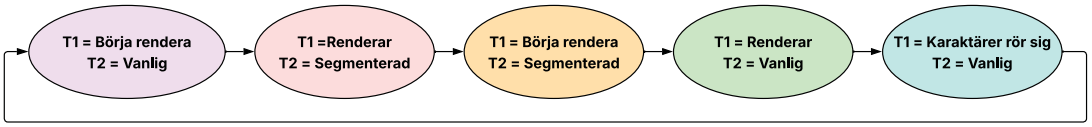
På liknande sätt skapas ett villkor hos renderingen och ett event hos karaktärerna som gör att renderingen inväntar att alla karaktärer har stannat, innan den börjar rendera.

5.2.3 Programflöde och synkronisering

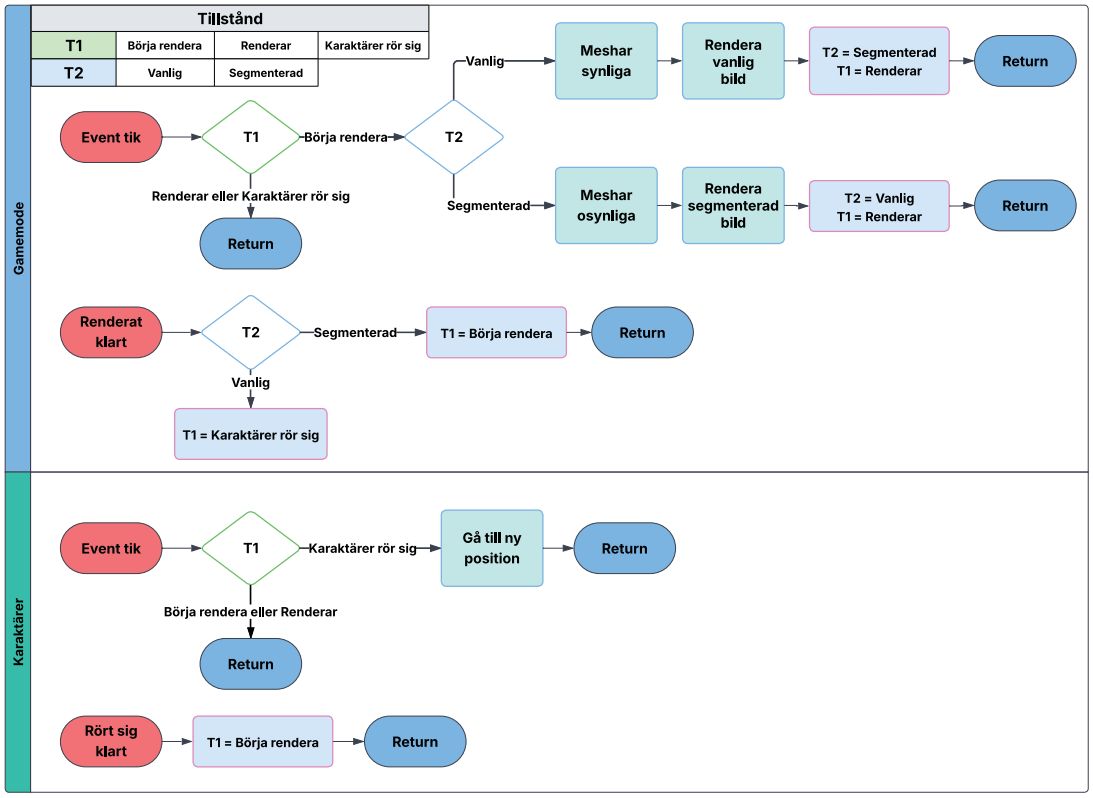


Figur 5.5: Figuren visar det sekventiella flödet för genereringen. Först flyttas karaktärerna till en slumpmässig position. Därefter vill man se till att alla meshar är osynliga innan man renderar den vanliga bilden. Därefter görs mesharna synliga, den segmenterade bilden renderas och processen upprepas.

Genereringen har ett sekventiellt programflöde enligt figur 5.5. Först flyttas karaktärerna till en slumpmässig position. Därefter vill man se till att alla meshar är osynliga innan man renderar den vanliga bilden. Därefter görs mesharna synliga, den segmenterade bilden renderas och processen upprepas. För att skapa det programflödet krävs synkronisering eftersom karaktärernas rörelse, bildrenderingen och huvudprocessen gamemode, exekveras parallellt med varandra. Processen byggs därför upp som en tillståndsmaskin med två tillstånd som visualiseras i figur 5.6 och 5.7.



Figur 5.6: Figuren visar övergången mellan tillstånden som användes för att synkronisera renderingen, segmenteringen och rörelsen av karaktärerna.



Figur 5.7: Flödesdiagram över genereringens styrlogik. Gamemode:ets och karaktärernas inbyggda klocka genererar kontinuerligt ett tick event som fungerar som kontrollslinga för hela processen. Två tillståndsvariabler används för att styra flödet: T1 och T2. T1 avgör om ett nytt renderingsjobb kan startas och växlar mellan tillstånden Börja rendera, Renderar och Karaktärer rör sig. T2 styr vilken typ av bild som renderas och kan anta tillstånden Vanlig eller Segmenterad.

Gamemode:ets inbyggda klocka ger kontinuerligt ut ett ”tick event” under simulationen och används därför som kontroll-loop och beror på två olika tillståndsvariabler T1 och T2. Tillståndet T1 styr om loopen kan starta ett nytt renderingsjobb eller inte och har tre olika tillstånd **Börja rendera**, **Renderar** och **karaktärer rör sig**. T2 kontrollerar vilken typ av bild som renderas och har två olika tillstånd **Vanlig** och **Segmenterad**. Likande Gamemode:et har alla karaktärer ett eget ”tick event” som används som loop, men denna styrs bara av T1.

Gamemodet börjar med att kolla om den ska börja rendera en bild genom att kontrollera tillståndet T1. Om tillståndet är **Börja rendera** går den vidare och kontrollerar T2 som bestämmer vilken bild som ska renderas. Om T2 är **Vanlig** görs mesharna osynliga och en vanlig bild renderas. Därefter sätts T2 till **segmenterad** och T1 till **Renderar** och ”tik eventet” retunerar. Då värdet på T1 är satt till **Renderar** kommer både Gamemodes och karaktärernas loop att retuneras direkt. Då renderingen är klar kallas automatiskt ett event som ändrar värdet på T1 beroende av värdet på T2. Eftersom T2 sattes till **Segmenterad** kommer T1 i detta

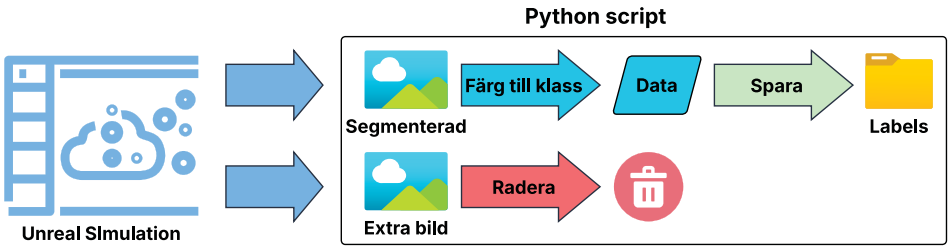
fall sättas till **Börja rendera**. Då kommer Gamemode:et komma förbi T1 och gå vidare till T2 där den denna gången kommer gå åt andra hållet. Mesharna görs först synliga sedan renderas den segmenterade bilden och därefter sätts T1 till **renderar** och T2 till **Vanlig**. När bilden har renderats klart kommer T1 sättas till karaktär rör sig och då kommer istället karaktärernas event loop att kunna gå vidare och karaktärerna rör sig till en ny position. När alla karaktärer nått sina positioner kallas ett event som sätter T1 till **Börja rendera** och tillståndsmaskinen har genomfört ett varv och ett resultat liknande figur 5.8 har erhållits.

Figur 5.8: Den vänstra bilden visar bilden som används som data medan den högra bilden visar motsvarande segmenterade bild som omvandlas till en segmenteringskarta.

5.2.4 Databearbetning

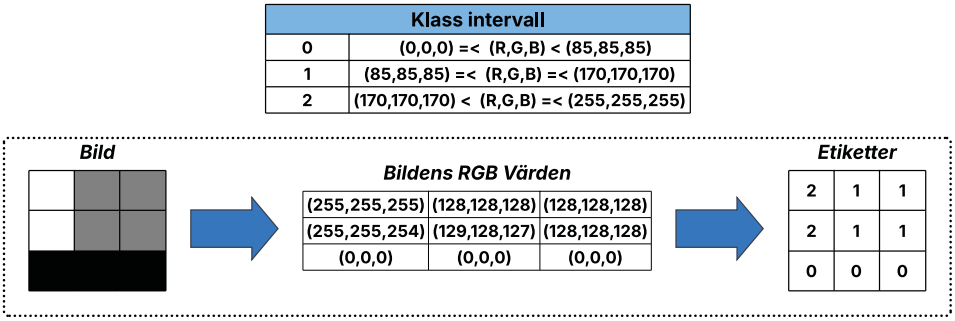
Den segmenterade bilden som har renderats behöver bearbetas eftersom den endast innehåller färger och inga klassetiketter. Dessutom renderas det alltid först en vanlig bild innan den segmenterade när man använder postprocessing material, vilket också behöver hanteras. Eftersom mesharna är synliga i bilden som renderas som en bieffekt, kan den inte användas och kan därför raderas, se Figur 5.9. För att filtrera ut och behandla bilderna används ett python script som visualiseras i figur 5.10. Python scriptet avläser mappen som ground-truth bilderna sparas i parallellt med att nya bilder skapas av UE5. Eftersom UE5 automatiskt ger de segmenterade bildernas filnamn prefixet stencil, kan det användas för att filtrera bilderna. Om filen som scriptet läser har stencil i filnamnet kommer den att bearbetas och om den inte har det kommer den att raderas.

Figur 5.9: Exempel på extra bild som genererats som konsekvens av postprocessing-materialet. Eftersom mesharna är synliga kan den inte användas och bör därför raderas.



Figur 5.10: Visualsering av python scriptet som raderar de överföldiga bilderna och omvandlar de segmenterade bilderna till segmenteringskartor.

För att hantera komprimeringar som skulle kunna påverka RGB-värdet skapas intervall för varje klass kring de färger som definierats i postprocessing-materialet i UE5. De definierade intervallen används för att transformera en matrisen med RGB värden till en matris med heltalvärden $k \in \{0, 1, \ldots, n-1\}$, där n är antalet klasser, se figur 5.11. Slutligen formateras matrisen om till en PNG-fil med endast gråskala, för att vara rätt format för semantiska segmenteringsmodellen som skapas i nästa kapitel.



Figur 5.11: Visualisering av hur bildens färger omvandlas till etiketter. För att hantera eventuella färgavvikelser orsakade av komprimering definieras intervall kring varje klasspecifik färg som används i postprocessing-materialet i UE5. Dessa intervall används för att transformera en RGB-matris till en heltalsmatris med värden $k \in \{0, 1, \ldots, n-1\}$, där n är antalet klasser.

5.3 Träning av semantiska segmenteringsmodeller

I detta avsnitt förklaras överföringsinlärning som sedan används för att träna en semantisk segmenteringsmodell på simulerad data.

5.3.1 Överföringsinlärning

Överföringsinlärning är en teknik inom maskinlärning där kunskap från en förtränad modell återanvänds för att träna en ny modell på en annan men relaterad domän. Detta är användbart inom bildigenkänning där de första lagren i en bildigenkänningsmodell lär sig att känna igen generella mönster som räta och horisontella linjer. Dessa mönster kombineras i de djupare lagren till mer komplexa strukturer, som i det sista lagret används för att göra en klassificering. Genom att ersätta klassificeringslagret med ett nytt som motsvarar de egna klasserna, samt frysa vikterna i övriga lager, kan modellen anpassas till de nya klasserna med en mindre mängd data än vad som hade krävts vid träning från grunden.

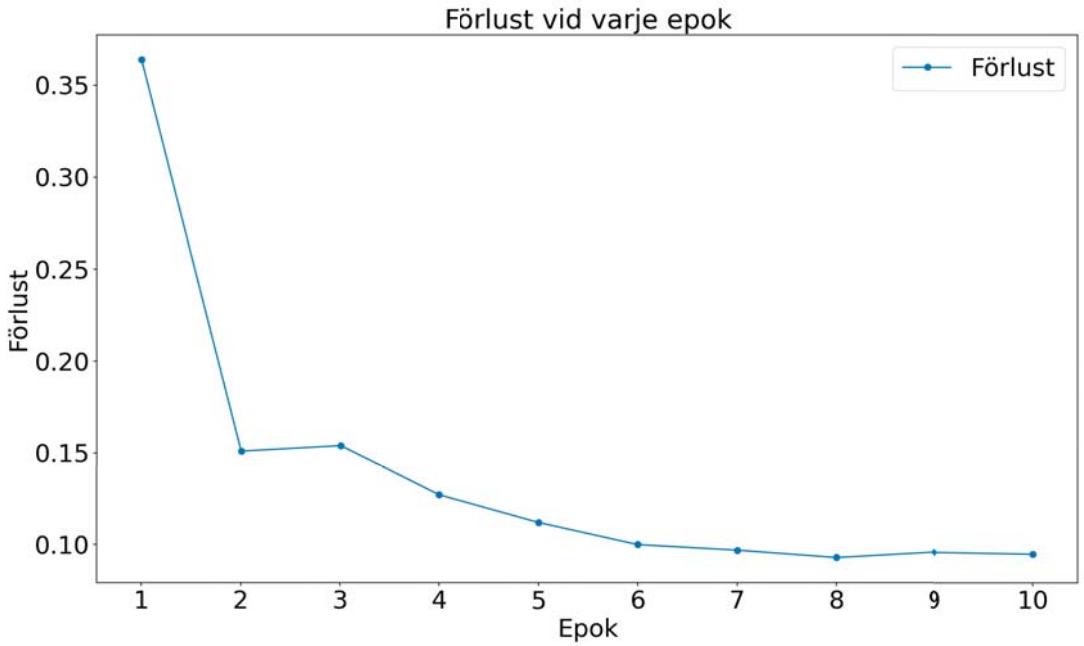
5.3.2 Träning på genererad data

För att utvärdera datakvaliteten tränas två semantiska segmenterings modeller på genererad ground-truth-data från den digitala tvillingen. Den första modellen har tre klasser ”robot”, ”golv” och ”hinder”. Klassen hinder inkluderar alla objekt som roboten inte kan köra igenom såsom hyllor, väggar och pallar. I den andra modellen kombineras klasserna ”golv” och ”hinder” till en klass ”inte robot”. Båda modellerna tränades och utvärderas på samma dataset bestående av 1500 bilder, som delats upp i 90% träningsdata och 10% testdata.

För att inte behöva träna modellerna från grunden används överföringsinlärning. Den förtränade modellen som används är pytorchs deeplabv3_resnet50 [33], vilket använder resnet-50 som backbone och deeplabv3 som klassificerare. Modellen är redan tränad på dataseten Pascal voc [34] och Coco[35] och kan segmentera 21 olika klasser. Genom att ersätta det sista konvolutionella lagret, vilket ansvarar för klassificeringen, till ett nytt konvolutionellt lager med samma antal klasser som den träningsdata som har genererats, kan modellerna tränas om för att känna igen klasserna i träningsdatan. På så sätt kan den färdigtränade modellens förmåga att identifiera generella mönster utnyttjas, vilket gör att det krävs mindre träningsdata för att träna modellerna på de nya klasserna. För att träna modellerna användes följande parametrar.

- CrossEntropyLoss valdes som förlustfunktion, då den är väl etablerad inom klassificeringsproblem och visat bra resultat i studier [36].
- Optimeringsalgoritmen Adam användes för att optimera de konvolutionella lagren, eftersom den har visat snabb och stabil konvergens i liknande applikationer [37].

- En inlärningshastighet av 0.001 valdes, vilket motsvarar standardvärdet hos Adam-optimeraren.
- Modellen tränades i åtta epoker, då förlustfunktionen visade tecken på konvergens vid denna punkt, se figur 5.12



Figur 5.12: En graph som visar modellens förlust över antalet epoker. Vid åtta epoker slutar förlusten att minska, vilket betyder att modellen har konvergerat.

Modellerna användes för att segmentera verkliga bilder från Volvo, som sedan användes för att utvärdera resultatet av en digital tvilling skapad med 3D Gaussian Splatting i nästa kapitel.

6
Resultat och diskussion

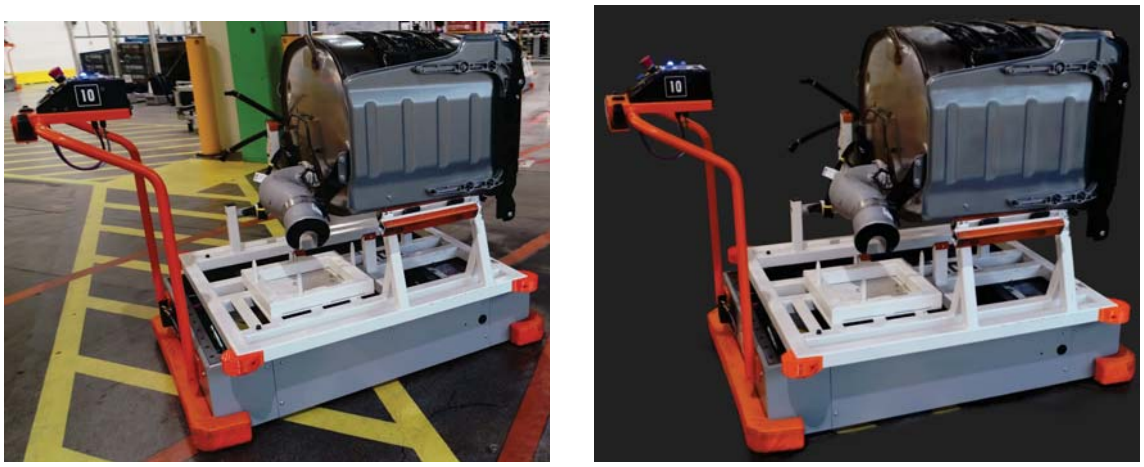
I detta kapitel presenteras och diskuteras arbetets resultat i tre olika huvuddelar. Dessutom diskuteras eventuella användningsområden, framtida utveckling och etiska aspekter kopplade till arbetet.

6.1 3D Gaussian Splatting

I följande avsnitt presenteras och analyseras resultaten av de skapade 3DGS-modellerna, med fokus på både omfattande miljöer och enskilda objekt.

6.1.1 3DGS-objekt i Postshot

Jämförelse mellan fotografi (a) och 3DGS-modell (b) i figur 6.1 visar att 3DGS-modellen av roboten lastad med ljuddämpare har klara likheter med verkligheten. Detaljer som numret på kontrollenheten syns tydligt i 3DGS-modellen och färgerna stämmer bra överens med fotot. Små förluster av detaljer kan ses på reflekterande ytor som toppen av ljuddämparen i figur 6.1 och resulterar i grumligare ytor. Detta kan möjligen bero på att optimeringen av 3DGS-modeller bygger på en jämförelse mellan den renderade modellen och de ursprungliga bilderna. Reflekterande ytor kan uppvisa variationer i utseende beroende på observationsvinkel, vilket leder till avvikelser mellan bilderna av samma yta. Dessa variationer försvårar optimeringsprocessen, då olika vinklar genererar motstridiga direktiv för hur modellen bör justeras. För fler exempel på 3DGS-objekt, se bilaga A.3 samt A.4.

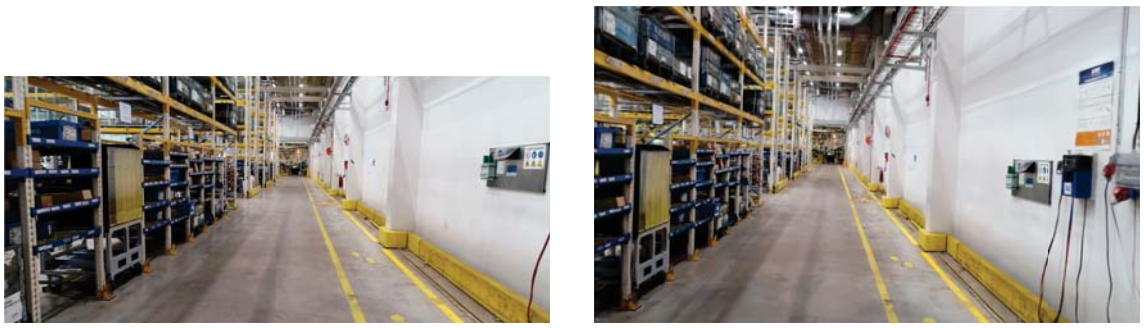


(a) Fotografi (b) 3DGS-modell i Postshot.

Figur 6.1: Jämförelse mellan verkligheten och 3DGS-modell i Post av en robot lastad med en ljuddämpare. Klara likheter syns mellan verkligheten och 3DGS-modellen.

6.1.2 3DGS-miljöer i Postshot

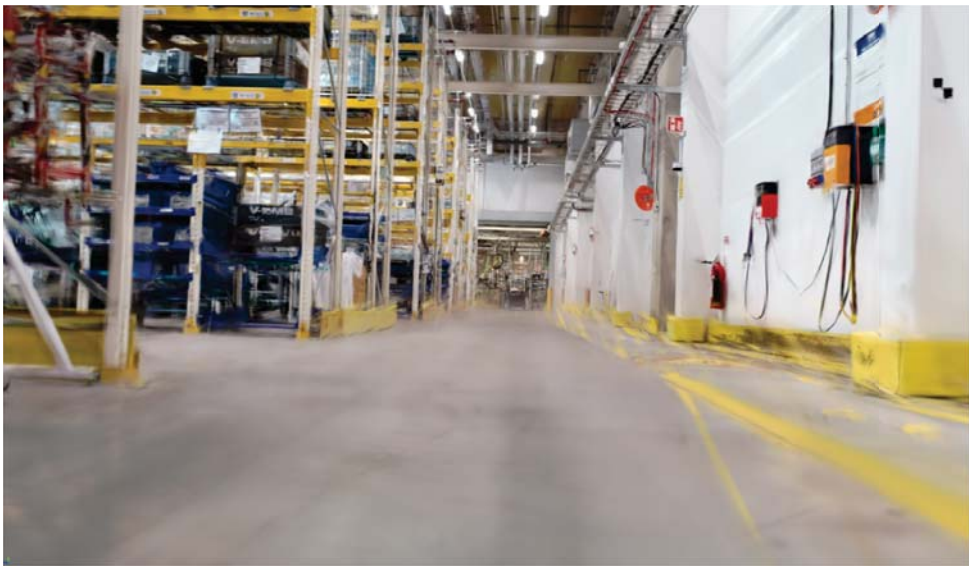
En jämförelse mellan fotografiet i figur 6.2(a) och 3DGS-modellen i figur 6.2(b), som visualiserar sektion 1, visar en hög grad av överensstämmelse. Samtliga detaljer återges korrekt i modellen. Mindre avvikelser, såsom färgåtergivning och ljussättning, kan observeras men påverkar inte modellens funktionalitet eller noggrannhet i någon större utsträckning.



(a) Fotografi (b) 3DGS-modell

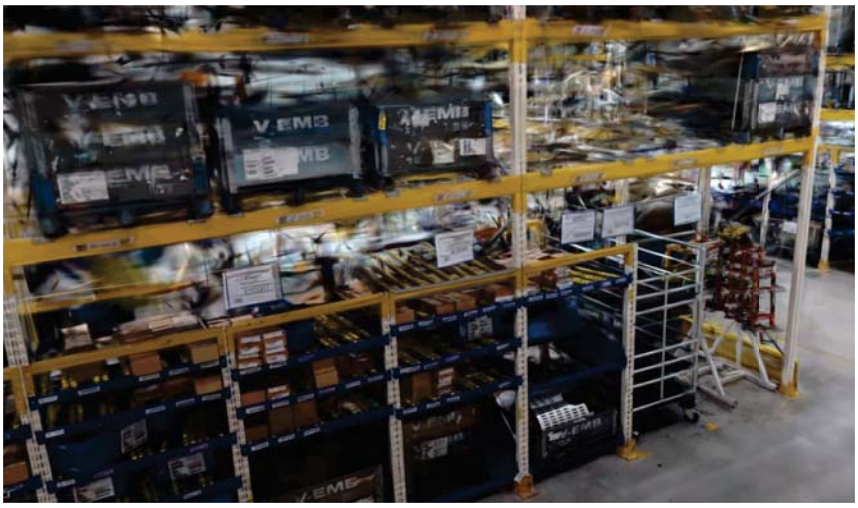
Figur 6.2: Visualisering på verkligheten samt 3DGS-modell av sektion 1 där endast små förändringar i färg och ljussättning kan urskiljas.

I figur 6.3 illustreras hur golvet i 3DGS-modellen inte är helt slätt, utan hur 3D-gaussians sticker upp ur ytan och skapar oregelbundenheter. På grund av saknad bilddata från vissa vinklar, exempelvis som i figur 6.3, finns det inget som styr optimeringen mot att rekonstruera ett helt plant golv.



Figur 6.3: Golvet i 3DGS-modellen av sektion 1 på AB Volvos fabrik i Tuve visar hur 3D-gaussians sticker upp från marken och skapar ett dimmigt golv.

Följande figur 6.4 visar hur nedre delen av hyllorna i sektion 1 har rekonstruerats med klara detaljer medan den övre delen är förvrängd. Samma fenomen förekommer även på de vita väggarna på motsatt sida av gången. Detta är på grund av bristande bilddata och är en konsekvens av den bildtagningsmetod som beskrevs i avsnitt 3.2.3. Dock förekommer det endast när hyllorna eller väggarna observeras rakt framifrån. Figur 6.2(b) visar att observationer längs med väggarna inte uppvisar dessa förvanskningar då bilddata från liknande vinklar som exempelvis 6.2(a) fanns tillgängligt. Detta samt golvet i figur 6.3 understryker att den bildtagningsmetod som använts på Volvo var bristfällig och behöver utvecklas om en fullständig rekonstruktion av miljön ska lyckas.

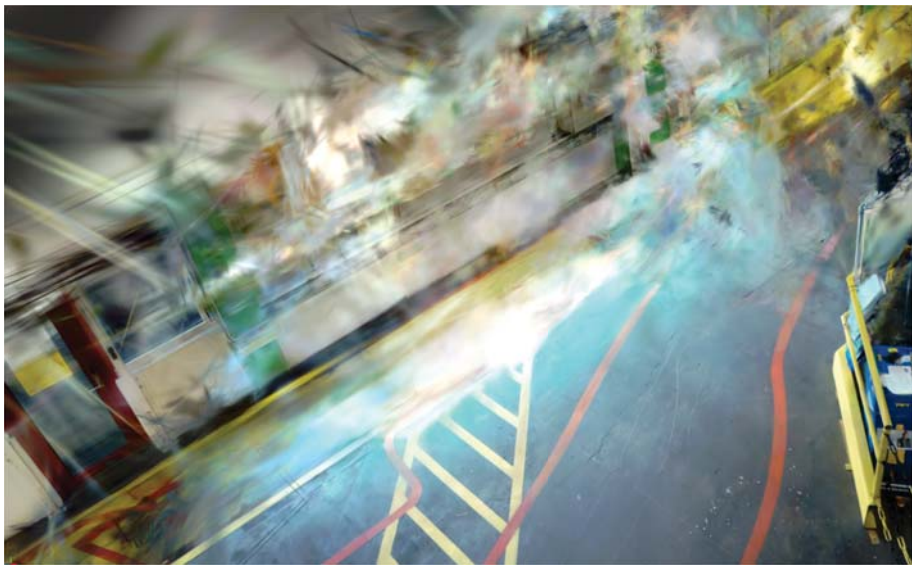


Figur 6.4: Visualisering av 3DGS-modell för hyllor i sektion 1. Den nedre delen är väl rekonstruerad med tydliga detaljer, medan den övre delen uppvisar förvrängningar, vilket indikerar bristfällig bilddata från höga vinklar. Modellen illustrerar hur begränsningar i datainsamlingen påverkar rekonstruktionskvaliteten i olika delar av miljön.

Figur 6.5 visar en jämförelse mellan ett fotografi (a) och en 3DGS-modell (b) av sektion 2. Till skillnad från sektion 1, där modellen uppvisade en hög grad av överensstämmelse, är skillnaderna lite mer påtagliga. Den genererade 3DGS-modellen uppvisar suddiga delar, exempelvis tavlorna och området där solljuset som tränger in genom fönstren, vilket illustreras i figur 6.5(a). Solljusets effekt framträder ännu tydligare i figur 6.6 och resulterar i suddiga 3D-gaussians kring fönstret. Detta tyder på att datainsamlingen för denna sektion varit mindre optimal, där faktorer som felaktig kamerapositionering och komplexa ljusförhållanden resulterat i otillräcklig bilddata. Tydliga strukturella drag kan fortfarande urskiljas i modellen, men inte med samma precision som i sektion 1. Detta understryker vikten av högkvalitativ datainsamling och optimala förhållanden vid skapandet av 3DGS-modeller.

(a) Fotografi **(b)** 3DGS-modell

Figur 6.5: Visualisering av verkligheten och motsvarande 3DGS-modell av sektion 2. Endast mindre skillnader kan urskiljas mellan bilderna, där modellen uppvisar något sämre skärpa i tavlan och i området kring solljuset.



Figur 6.6: Visualisering av ljusförhållanden och dess påverkan på 3DGS-modellen av sektion 2 i Volvos Tuvefabrik

Figur 6.7 visar en jämförelse mellan ett fotografi (a) och den motsvarande 3DGS-modellen (b) av sektion 3 i PostShot. Precis som i sektion 2 liknar 3DGS-modellen fotografiet, men vissa avvikelser kan identifieras. Svävande 3D-gaussians syns runt det svarta bordet, och små detaljer, såsom ytan på det silvriga området till höger, går förlorade. Figur 6.8 visar samma modell som i figur 6.7, men betraktad från en vinkel som inte fanns representerad i det bildmaterial som användes vid modellträningen. Från detta perspektiv framstår modellen som betydligt sämre. Golvet är i stora drag igenkännbart, men väggarna är diffusa och stora delar av sektionen blockeras av svarta 3D-gaussians. Denna bristande modell förklaras troligen av otillräcklig bilddata från varierande vinklar, vilket ytterligare bekräftar tidigare påståenden om att bildtagningsmetoden varit bristfällig och behöver förbättras.

(a) Fotografi **(b)** 3DGS-modell

Figur 6.7: Visualisering av verkligheten samt 3DGS-modell av sektion 3. Små skillnader kan ses i form av svävande 3D-gaussians och förlorade detaljer på ytor.

Figur 6.8: 3DGS-modell av sektion 3 i PostShot, observerad från en vinkel som ej fanns med i bilddata, vilket resulterade i en försämrad rekonstruktion.

Flera 3DGS-modeller av miljöer och objekt visar att det generellt är enklare att skapa en högkvalitativ 3DGS-modell av ett objekt än av en miljö. Detta beror främst på att optimal fotograferingsteknik är lättare att tillämpa på objekt samt att den större skalan hos miljöer ökar risken för att viktiga vinklar missas vid inspelning. Ett alternativt sätt att se på problemet är att förenkla en miljö till en uppsättning objekt. För bra rekonstruktion av ett objekt behövs bilddata runt hela objektet för att säkerställa bra täckning. För bra rekonstruering av miljön behövs därför denna 360-graders täckning för varje objekt i miljön. Detta förklarar varför antalet bilder som behövs till miljöer är betydligt fler, särskilt när miljöerna blir mer komplexa. Dessutom kan fotograferingstekniken av miljöer begränsas av miljön själv då väggar, hyllor och andra hinder kan försvåra datainsamlingen.

6.1.3 3DGS-objekt i UE5

En jämförelse mellan (a) och (b) i figur 6.9 visar att utseendet på 3DGS-modellen i UE5 av robotkaraktären är mycket lik sin motsvarighet i Postshot. Dock är den importerade 3DGS-modellen inte en exakt kopia. Figuren visar att modellens visuella kvalitet försämras vid import till UE5 i förhållande till den renderade modellen i Postshot. Denna försämring kan ses vid jämförelse mellan figur 6.9 (a) och figur 6.9 (b). Modellen i UE5 är något mörkare än verkligheten och motsvarande modell i Postshot. Notera att modellen som visas i figur 6.9 (b) inte tar hänsyn till det simulerade ljuset i UE5. Den dynamiska modellen, vilket tar hänsyn till ljuset i UE5-miljön gör detta problem redundant eftersom ljusstyrkan höjs till en nivå som överensstämmer med verkligheten. Dessutom är hela den silverfärgade ytan på ljuddämparen, som är lastad på roboten, något fläckig vilket inte förekommer i verkligheten eller av samma magnitud i Postshot. Detta beror troligtvis på begränsningar i hur grafiken representeras då den importeras till UE5.

(a) Bild på 3DGS-modell i Postshot. **(b)** Bild på 3DGS-modell i UE5.

Figur 6.9: Jämförelse mellan Postshot (a) och UE5 (B) på 3DGS-modellen av en robot lastad med ljuddämpare. Ytorna på ljuddämparen blir fläckigare och modellen blir mörkare.

Utöver att modellen är något mörkare och fläckig består skuggans skärpa vid importering till UE5. Detta visas på robotens ljusgråa sida, under den vita ramen. Ytterligare detaljer som kvarstår efter importering är till exempel den orangea ramen och handtaget samt dekalen ”10” på kontrollenheten i det övre vänstra hörnet, vilket överensstämmer med resultatet som presenterades i avsnitt 6.1.1.

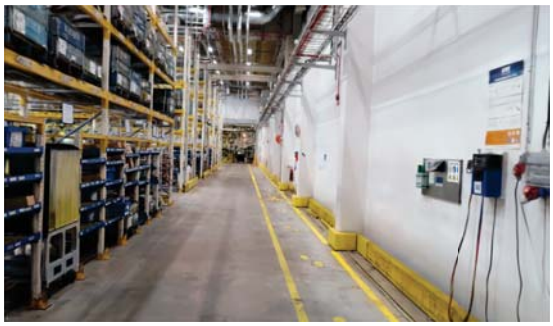
För ytterligare jämförelse mellan objekt i Postshot och UE5, se bilagorna A.3 och A.4. Där kan liknande slutsatser dras kring representationen i UE5. Till exempel är gaffeltruckens interiör betydligt mörkare samt den gråa ytan något fläckig. Gaffeltrucken samt roboten visar dock att skärpan kvarstår då det går att tyda dekalerna ”1381” (nedanför hytten) samt ”15” (på kontrollenheten) på respektive objekt.

Ur ett helhetsperspektiv ser 3DGS-modeller av roboten mycket verklighetstrogna ut men de har en del oönskade defekter som uppstår vid importeringen till UE5.

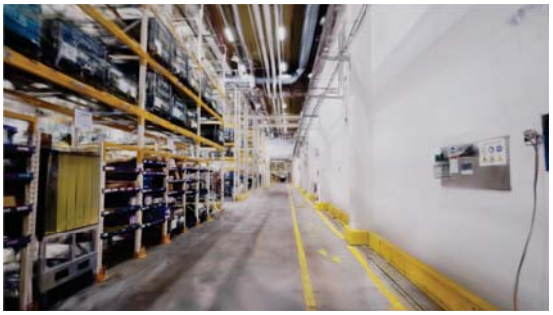
6.1.4 3DGS miljö i UE5

I figur 6.10 syns en jämförelse av sektion 1 i Postshot(a) och UE5(b). Modellen har betydligt sämre kvalitet i UE5 än i Postshot, då den uppvisar låg skärpa. Om man kollar i övre vänstra hörnet ser man hur pallarna har betydligt sämre kvalitet och att de nästan är genomskinliga. Detta tyder på att modellen blir komprimerad i UE5 och att den använder färre 3D-gaussians, vilket blir extra problematiskt för delar av modellen som redan har få 3D-gaussians. Delar som till exempel golvet som hade hög kvalitet i Postshot, påverkas inte lika mycket. Genom att titta på figur 6.11 ser man tydligt hur 3D-gaussians har försvunnit och att delar av pallarna försvinner. Detta skiljer sig markant från när roboten importerades till UE5, då kvalitet fortfarande var bra och den största förändringen var i ljuset. Detta visar, precis som tidigare

resultat i avsnitt 6.1.2, att det är lättare att göra 3DGS-modeller av objekt än av omgivningar.



(a) 3DGS-modell i Postshot



(b) 3DGS-modell i UE5

Figur 6.10: Visualisering på 3DGS-modellen av sektion 1 i Postshot (a) och UE5 (b). I (b) syns defekter form av minskat antal gaussians uppe i det vänstra hörnet. Överlag framstår modellen generellt suddigare i UE5 än i Postshot, vilket tyder på att gaussians har optimerats bort.

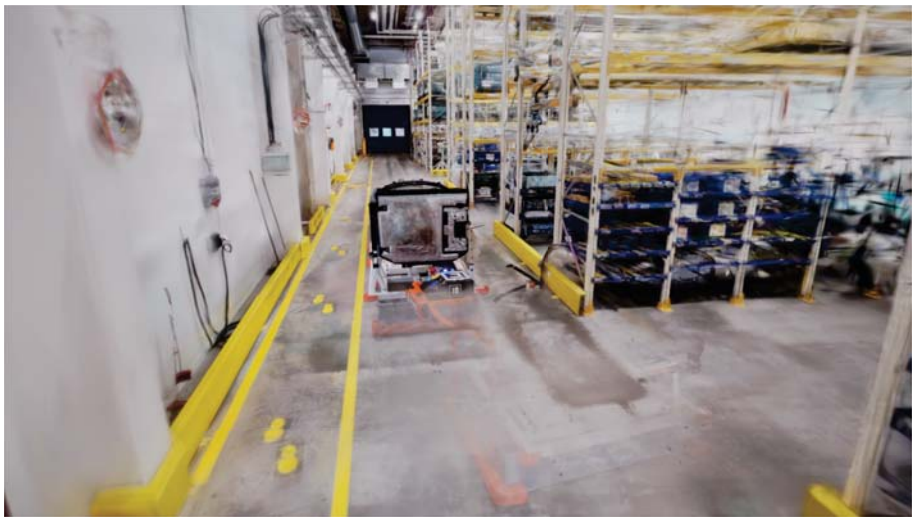
(a) 3DGS-modell i Postshot

(b) 3DGS-modell i UE5

Figur 6.11: Jämförelse av 3DGS-modeller av hyllor i sektion 1, visualiserade i Postshot (a) och UE5 (b). Modellen från Postshot uppvisar högre detaljrikedom och skärpa, särskilt i den nedre delen av hyllorna, medan modellen i UE5 är mer suddig och tappar detaljer, vilket tyder på kvalitetsförlust vid modellöverföring eller rendering i UE5.

När man sätter samman flera 3DGS-modeller i UE5 förekommer ibland problem som i figur 6.12. I bilden finns två robotar en närmare som nästan är osynlig och en lite längre bort där underdelen är dimmig. Att roboten längst bort är dimmig, hade kunnat förklaras med resultatet i avsnitt 6.1.2 som visar hur golvet inte är solitt utan består av en dimma av 3D-gaussians. Några av dessa 3D-gaussians hamnar då framför robotens nedre del, vilket gör att den ser dimmig ut. För roboten närmast finns inga 3D-gaussians framför den, vilket tyder på att det är ett problem i UE5:s renderingen, där det uppstår osäkerhet kring vilket objekt som är närmast och bör renderas. Detta är förmodligen en bugg hos tredjepartstillägget och förväntas åtgärdas i framtiden, antingen genom att tillägget förbättras eller genom att UE5 får

stöd för 3DGS-modeller. Även om resultatet i detta avsnitt inte var optimalt och flera brister har redovisas, är det viktigt att poängtera att det även finns bra vinklar som gör modellen användbar, vilket visas i avsnitt 6.3.



Figur 6.12: Figuren visar problem som kan uppstå när flera 3DGS-modeller kombineras i UE5. Den bortre robotens nedre del ser dimmig ut på grund av golvets 3D-gaussians som skymmer den. Den främre roboten är nästan osynlig, troligen på grund av ett renderingsfel i UE5 där djupordningen hanteras fel.

6.2 Simulering

Här presenteras och diskuteras resultaten kopplat till simuleringen i Unreal Engine 5. Resultatet för de digitala karaktärer bygger på dess visuella representation samt dess agerande jämfört med verkligheten. Ytterligare undersöks resultatet för störningar och anomalier samt fördelarna detta medför i simulationen.

6.2.1 Digitala karaktärer i UE5

Implementationen av digitala karaktärer består av två delar. Den visuella delen som innehåller utseende och mesh, samt beteende-delen och hur karaktären förflyttar sig i miljön. Karaktärernas utseende tas upp i avsnitt 6.1 medan resterande delar presenteras nedan.

En karaktärs mesh är som tidigare beskrivet i avsnitt 4.3 en volymrepresentation av ett objekt. I ett perfekt scenario betyder detta att en mesh överensstämmer helt med den grafiska modellen. Detta är inte fallet eftersom figur 6.13 visar några detaljer som inte överensstämmer. Ett exempel är den puckformade detaljen ovanför stötfångaren i det nedre högra hörnet. Den orangea delen från 3DGS-modellen sitter något höger om den gråa delen från mesh-objektet. Detta beror på att 3DGS-modellen och mesh-objektet är skapade utifrån två olika punktmoln där det ena är genererat med Postshot och det andra med RealityCapture. Detta kan ge små skillnader i position och skala vilket visas i figuren. Däremot med avseende på ändamålet att ge objektet en volymrepresentation är dessa få och små olikheter obetydande för att karaktären ska ge en god representation av verkligheten.

Figur 6.13: Sammanställd karaktär med 3DGS modell och mesh som förtydligar positions- och skalningsdefekterna mellan objekt skapade från två olika punktmoln. Den röda cirkeln visar hur positionen, för den orangea detaljen, hos 3DGS-modellen inte överensstämmer med samma detalj i mesh-objektet.

Slutimplementationen består av två olika beteenden som en karaktär kan tilldelas. Det ena bygger på att karaktären slumpmässigt väljer ut ett mål i miljön att färdas till. När karaktären väl har nått målet väljer den därefter ett nytt slumpmässigt mål. Det andra beteendet bygger på att karaktären följer en förutbestämd bana och upprepar denna i all oändlighet. I miljön finns utplacerade markörer som tilldelas till en karaktär. Karaktären färdas därmed mellan dessa markörer i den tilldelade ordningen.

Det finns både för- och nackdelar med dessa implementeringar. Det slumpmässiga beteendet är, intuitivt sett, inte verklighetstroget. Detta eftersom exempelvis robotar som ska transportera saker i en fabrik aldrig kommer lyckas att konsekvent nå sin slutdestination genom att slumpmässigt välja ett mål i omgivningen. Därför är det andra beteendet en mycket bättre representation av verkligheten. Dock har även detta beteende vissa begränsningar. Det tar till exempel inte hänsyn till om något är i vägen för karaktären då den förflyttar sig enligt banan. Karaktären kommer alltså kollidera med objekt som befinner sig i vägen och kan på så sätt fastna.

Liknande händelser kan även inträffa för det slumpmässiga beteendet men där väljs istället ett nytt slumpmässigt mål om karaktären fastnar och på så sätt inte når sitt mål inom en viss tid.

De främsta fördelarna med de två olika implementationerna är kopplade till datagenereringen för träningen av en AI-modell vilket tidigare nämnts i Kapitel 5. De två olika beteendena ger möjligheten att generera kontinuerlig verklighetstrogen data men med begränsad variation då karaktären endast kan befinna sig längs med banan. Det slumpmässiga beteendet säkerställer på så sätt en diversifierad datamängd genom att komplimentera med varierad data.

6.2.2 Simulerade störningar och ovanliga situationer

För att undersöka potentialen i den skapade digitala tvillingen har simuleringar av olika störningar och ovanliga situationer genomförts.

Simuleringen av samtliga störningar i UE5, förutom ljusskillnader, gjordes med olika Dirt Masks, vilket beskrevs i avsnitt 4.4.2. Figur 6.14 visar resultatet av en bildtagning med en Dirt Mask som simulerar en linsöverstrålning. Ytterligare bilder med simulerade störningar finns i bilaga A.9.

Figur 6.14: Simulerad linsöverstrålning med hjälp av en Dirt Mask-textur i Unreal Engine 5 (UE5). Bilden visar hur en artificiell smutsmask appliceras för att efterlikna optiska störningar i kameralinsen och öka realism i den digitala miljön.

Trots ett stort urval av inbyggda Dirt Masks-texturer och möjligheten att skapa egna är lösningen begränsad i sin variabilitet. De imperfektioner som appliceras på kameralinserna är statiska. Det medför att alla bilder som är renderade med samma lins uppvisar identiska störningar. En konsekvens är att resultatet inte blir lika verklighetstroget, eftersom damm, ljusreflektioner och spindelnät i verkligheten varierar i utseende och placering. Utöver försämrad realism kan bilderna potentiellt påverka en maskininlärningsmodells precision negativt om de används som träningsdata. Denna brist på variation i de simulerade störningarna innebär en risk för att en maskininlärningsmodell, om den tränades på sådana data, skulle överanpassas till just

dessa statiska störningsmönster istället för att lära sig generalisera till verkliga, varierande förhållanden. Det är dock viktigt att poängtera att syftet med bilderna i detta arbete är att fungera som ground-truth-data för utvärdering och demonstration av metoden. Bilderna är således inte optimerade för träning av maskininlärningsmodeller. Alternativa metoder för simulering av störningar kan med fördel utforskas, eftersom ett varierat och trovärdigt resultat är att föredra. Ett exempel skulle kunna vara att simulera damm som virvlar runt i rummet.

Utöver störningar simulerades även ovanligt förekommande situationer, vilket redogörs för i avsnitt 4.4.3. Figur 6.15 visar ett exempel på en simulerad brand i den digitala tvillingen. Bild (b) visar den tillhörande strukturella informationen. Bild (a) illustrerar den visuella representationen av elden i en soptunna, som i sitt nuvarande skick framstår som visuellt förenklad och saknar den detaljrikedom och realism som Unreal Engine 5 är kapabel att leverera. Detta exempel visar dock att det är möjligt att generera en visuell återgivning av brand. Denna initiala visualisering utgör en startpunkt, och en uppenbar utvecklingsmöjlighet är att utnyttja UE5:s verktyg för att skapa betydligt mer övertygande bränder vilket potentiellt kan förbättra träningen av maskininlärningsmodeller. Simuleringen illustrerar hur ovanliga och potentiellt farliga situationer kan visualiseras i den digitala tvillingen. Detta understryker värdet av en virtuell miljö för att studera och förstå sådana händelser.

(a) Visuell representation av simulerad eld

(b) Strukturell information om simulerad eld

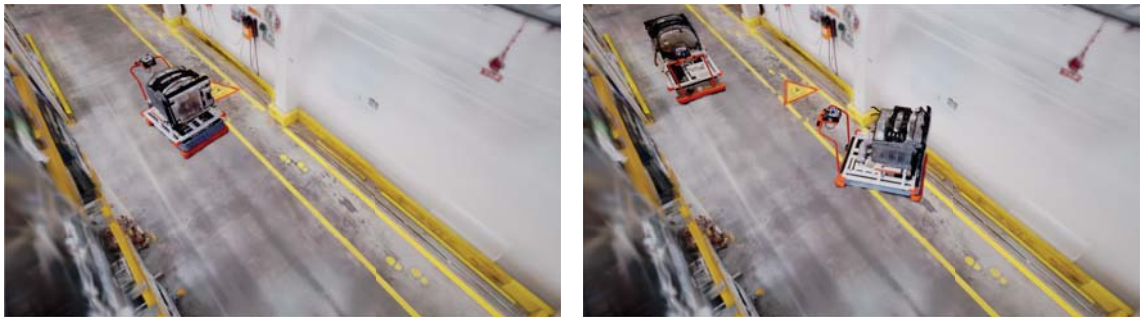
Figur 6.15: Simulering av brand i UE5 med visuell och strukturell representation.

6.3 Segmentering och AI

I detta avsnitt redovisas och diskuteras den ground-truth-data som genererades i simuleringen, samt hur den AI modell som tränats på simulerad data presterar. Avslutningsvis presenteras även möjliga förbättringar.

6.3.1 Utvärdering av genererat dataset

Det har genererats ground-truth-data ifrån tre olika kameravinklar, som syns i figur 6.16 och figur 6.17. Från varje vinkel har 500 bilder generats där varje bild är unik eftersom truckarna rör sig slumpmässigt. Antalet truckar i simuleringen är konstant två men eftersom de kan röra sig utanför kamerornas synfält, kan en bild innehålla en, två eller inga truckar.



(a) Vinkel 1 med en robot **(b)** Vinkel 1 med två robotar

Figur 6.16: Figuren visar exempel på unik träningsdata med olika antal robotar ifrån vinkel 1.

(a) Vinkel 2 **(b)** Vinkel 3

Figur 6.17: Figuren visar exempel på genererad träningsdata från vinkel 2 (a) och vinkel 3 (b).

Det enda som är statiskt i bilderna är kameravinklarna, vilket innebär en risk att modellen överanpassas till dessa vinklar. Därför hade det varit fördelaktigt att använda fler kameravinklar, men den bristande kvaliteten på 3DGS-modellen i UE5,

som redovisades i avsnitt 6.1.2, begränsar antalet vinklar med tillräckligt god kvalitet. Även i de kameravinklar som används förekommer tydliga brister, till exempel i vänstra hörnet i vinkel 2. Detta påverkar resultatet av en tränad modell och särskilt i hur domänanpassad den blir till Volvos lokaler.

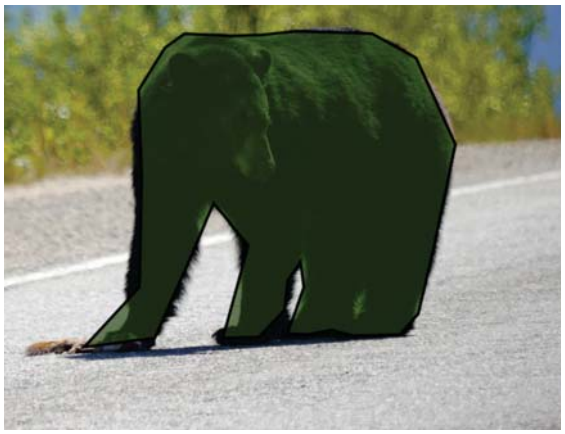
Figur 6.18(a) visar resultatet av segmenteringen från den digitala tvillingen. Segmenteringen är detaljrik och fångar även små detaljer, såsom robotens handtag. De enda bristerna i segmenteringarna utgörs av de små prickarna på roboten som syns i figur 6.18(b), vilket beror på att robotens mesh har små hål i sig. Vid en jämförelse med bilden i figur 6.18(a) och bilderna i figur 6.19 observeras en tydlig skillnad i segmenteringens noggrannhet mellan automatiskt annoterad data och manuellt annoterad data från datasetet COCO[35]. Segmenteringen i COCO-datasetet använder polygoner med raka kanter för att markera objekt, vilket resulterar i en förenklad bild av objektets form. Detta kan leda till felklassificeringar, exempelvis som i 6.19 (c) där en del av vägen är annoterad som klassen buss. Den förenklade formen hos segmenteringen är ett sätt att påskynda den manuella segmenteringsprocessen, som trots förenklingar tar flera sekunder per bild. Detta är avsevärt långsammare än segmenteringen i UE5, som teoretiskt kan renderas i 60 bilder per sekund tack vare spelmotorns optimerade grafikpipeline. I detta projekt begränsades dock renderingshastigheten av valet av renderingsverktyg, då bildtagning med Movie Render Queue tog två till tre sekunder per bild. En möjlig lösning skulle kunna vara att nyttja spelläget i UE5 för bildtagning, eftersom det renderar den uppbyggda simuleringen i över 60 bilder per sekund med de datorspecifikationer som redovisas i appendix A.10.

(a) Segmenterad bild på en robot från UE5.

(b) Inzoomad bild på en segmenterad robot, som visar på hål i segmenteringen.

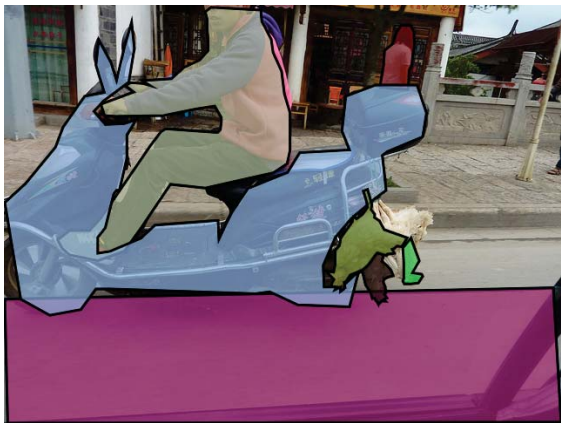
Figur 6.18: Figuren visar i (a) en segmenterad bild från UE5 där golvet (vitt), väggarna (svart) och en robot(blå) har segmenterats. Det förekommer Små vita prickar i den annars blå segmenteringskartan, vilket tydliggörs i den inzoomade bilden (b).

(a) Manuellt annoterad bild av en motorcykel och ett paraply från COCO



(b) Manuellt annoterad bild av björn från COCO

(c) Manuellt annoterad bild av en buss från COCO



(d) Manuellt annoterad bild av en man på en moped från COCO

Figur 6.19: Figuren visar fyra exempel på manuellt annoterade bilder från datasetet COCO[35]. Segmenteringen består av polygoner med raka kanter vilket gör att objektens form förenklas, vilket kan leda till felaktiga annoteringar. Exempel på fel är staketet bakom motorcykeln i (a) samt en del a vägen som klassas som buss i (c).

Även om majoriteten av data är korrekt och verklighetstrogen, så förekommer det defekter när genereringen sker automatiskt. I figur 6.20 syns två exempel på defekt data. I bild (a) har robotarna lyckats hamna på varandra, vilket är ett overkligt scenario. Detta visar att robotarnas beteenden kan förbättras. Däremot kommer det inte ha någon större påverkan på modellen eftersom bilden fortfarande är korrekt klassad. I bild (b) förekommer problemet med golvet och dess 3D-gaussians som diskuteras i avsnitt 6.1.4. Golvets 3D-gaussians hamnar ibland framför roboten, vilket gör att delar av roboten döljs. Segmenteringen tar dock inte hänsyn till 3D-gaussians, då den baseras på objektens meshar. Det innebär att hela roboten ändå syns i segmenteringskartan, trots att delar av den är skymda i originalbilden. Detta resulterar i felaktigt klassificerad data, eftersom det som framstår som en del av golvet i originalbilden kommer att klassas som en robot i segmenteringskar-

tan. Enstaka fel i annoterad ground-truth-data påverkar sällan modellens prestanda nämnvärt, men en ökad förekomst kan försämra prestandan.

(a) En robot står på den andra **(b)** Delar av en robot försvinner i golvet

Figur 6.20: Figuren visar exempel på genererad data som innehåller overkliga scenarion.I (a) har en robot hamnat ovanpå en annan robot och i (b) har delar av roboten försvunnit på grund av golvets gaussians.

6.3.2 Träning av semantisk segmenteringsmodell

En semantisk segmenterings modell har tränats på datasetet som redovisas i avsnitt 6.3.1. Datasetet har delats upp i en träningsdel och en testdel, enligt beskrivningen i avsnitt 5.3.2. Efter avslutad träning utvärderades modellerna på testdatasetet och resultaten erhölls presenteras i tabell 6.1 och tabell 6.2.

Utvärderingsmått	Tränad på 3 klasser	Tränad på 2 klasser
Pixelnoggrannhet	0.9943	0.9961
Genomsnittlig pixelnoggrannhet	0.9639	0.98
Genomsnittlig IoU	0.9418	0.9568
Genomsnittlig Dice noggrannhet	0.9686	0.9771

Tabell 6.1: Tabellen visar modellernas noggrannhet på testdatasetet utifrån 4 olika utvärderingsmått, Pixelnoggrannhet, Genomsnittlig pixelnoggranhet, Genomsnittlig IoU(Intersection over Union) och genomsnittlig Dice noggrannhet.

Utvärderingsmått	Hinder	Golv	Robot
IoU per klass	0.998	0.992	0.88
Dice noggrannhet per klass	0.999	0.996	0.936

Tabell 6.2: Tabellen visar IoU(Intersection over Union) och Dice noggrannheten per klass för modellen med 3 klasser.

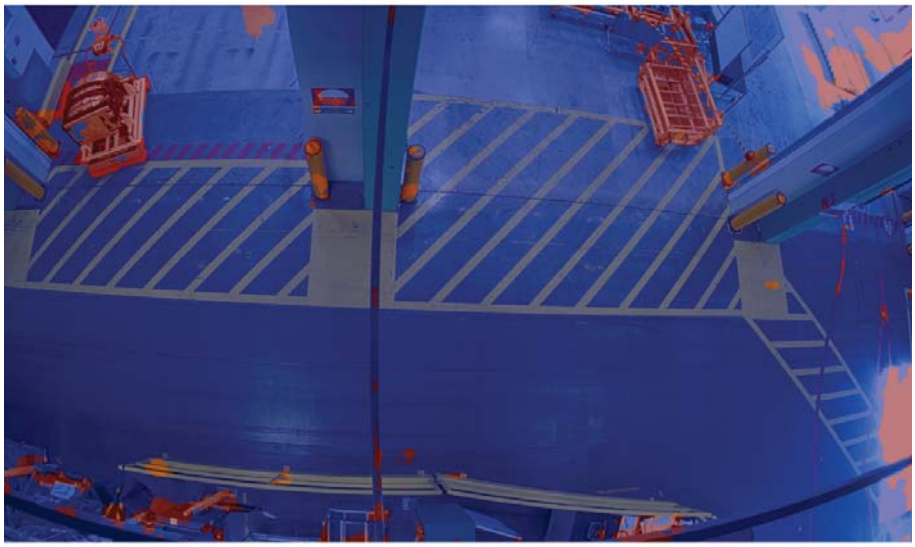
Modellerna visar hög noggrannhet med över 90 procent noggrannhet på alla mått förutom IoU för ”Robot” klassen som visar 84 procent vilket fortfarande är högt. Resultatet visar att modellerna nästan uppnår en perfekt klassificeringsnoggrannhet för klasserna ”hinder” och ”golv”. Detta kan förklaras av att kameravinklarna i test-

och träningsdatasetet är identiska vilket leder till att modellerna redan har sett golv och väggar innan. Det är också ett tecken på att modellerna kan ha blivit överanpassade till just de vinklarna, vilket kan bli problematisk om modellen används i andra vinklar. Att modellerna inte visar lika högprecision på klassen ”robot” är ett tecken på att den klassen inte har blivit överanpassad.

6.3.3 Klassificering av verkligheten

De två AI-modellerna med två respektive tre klasser har segmenterat sju bilder ifrån olika kameror inuti Volvos fabrik i Tuve. Bilderna är unika och innehåller varierande antal robotar från noll till tre per bild. Sammanlagt förekommer det åtta robotar i alla bilder. Ingen av modellerna har inte tränats på någon av vinklarna som förekommer i de verkliga bilderna. Vilket gör att de inte kan dra nytta av en eventuell överanpassning där de har lärt sig var i bilderna golvet, väggarna och robotarna befinner sig för att ge ett falskt resultat.

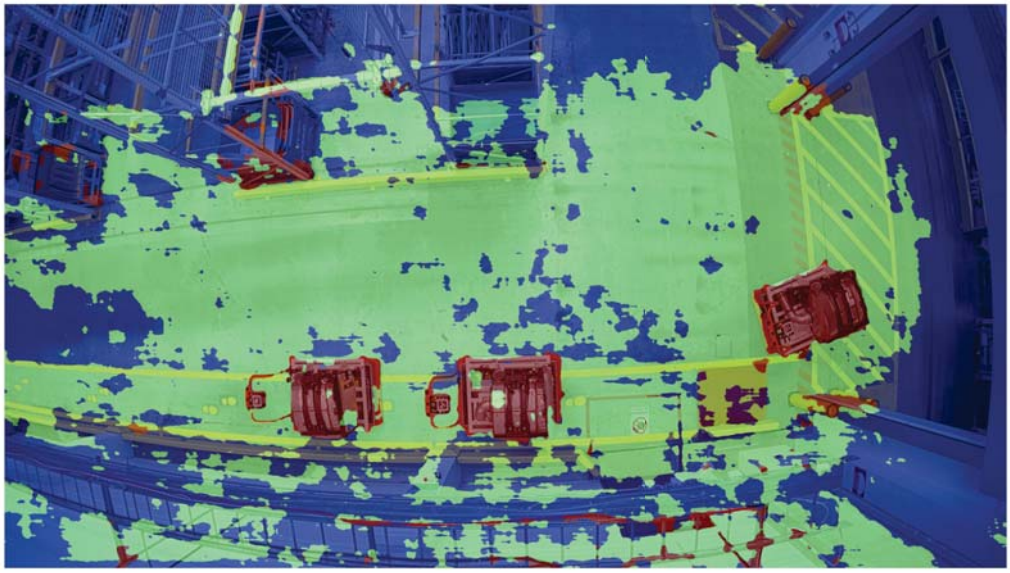
Figur 6.21 visar ett resultat av klassificering av en verklig bild från fabriken i Tuve. Bilden består av två klasser där röd är ”robot” och blå ”inte robot”. I detta exempel framgår det att AI-modellen lyckas identifiera roboten som befinner sig i det övre vänstra hörnet och segmenterar objektet med tydliga konturer. Dessvärre klassificerar den även några få små områden av, till exempel pelare och golvet som ”robot” men även vagnen som befinner sig i det övre högra hörnet. Detta kan bero på att vagnen är lik roboten samt att den inte har ingått simuleringen tillsammans med segmenteringsalgoritmen. Detta visar att modellen inte fått den data som krävs för att urskilja objekt som är lika roboten och det hade krävts mer data på sådana objekt. Ytterligare felklassificeringar visas på roboten där några fläckar är klassade som ”inte robot”. Felklassificeringarna på roboten kan vara en konsekvens av att träningsbilderna på roboten inte varit tillräckligt verklighetstrogna. En bidragande faktor kan vara att endast en typ av belysning användes i samtliga bilder.



Figur 6.21: Visar en segmenterad bild, från AB Volvos fabrik i Tuve, med två klasser ”robot” (röd) och ”inte robot” (blå). Semantiska segmenterings-modellen har lyckats segmentera roboten i stort sett korrekt med endast små defekter.Vissa objekt har dock klassificerats felaktigt som ”robot”, till exempel vagnen i övre högra hörnet.

Fler bilder som klassificerats med två klasser presenteras i bilaga A.5 där resultatet

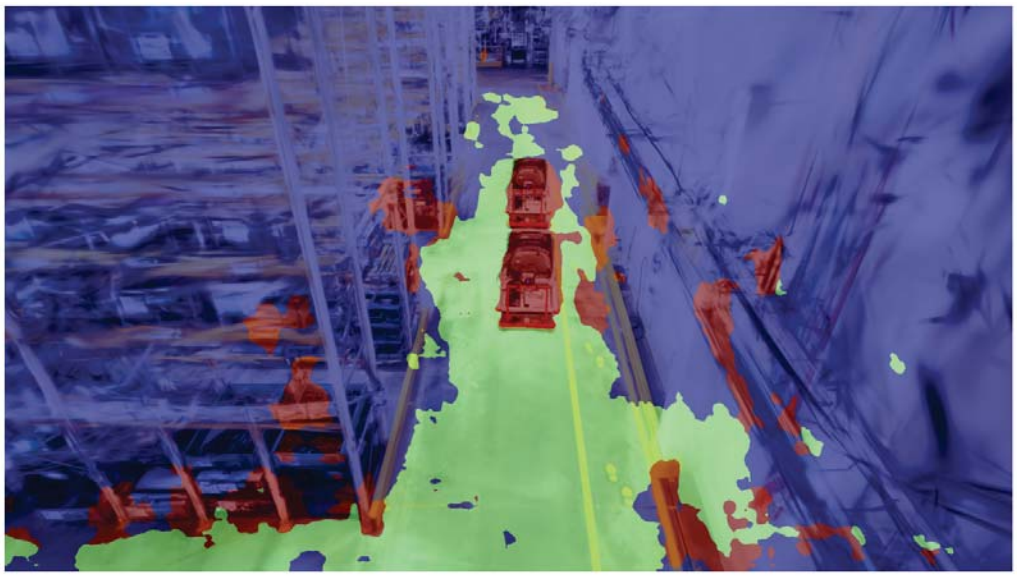
är mycket likt det i figur 6.21. Bilderna visar att modellen klarar att klassificera robotarna och omgivningen väl med några undantag. Golvet och väggarna klassas främst som ”inte robot”, medan vissa objekt så som pelare, kablar, gaffeltruckar och förvaringshyllor felklassificeras som ”robot”. Detta visar ytterligare tecken på att den data som använts vid träning av modellen består av för lite variation. Datavariationen är begränsad eftersom att simuleringen inte innehöll några dynamiska objekt som var klassade som ”inte robot”. Detta leder till att de enda ”inte robot” objekten modellen tränats på är de tre vinklarnas bakgrunder vilka är statiska i alla bilder.
Figur 6.22 visar resultatet från segmenterings-modellen med tre klasser, ”golv” (grön), ”hinder” (blå) och ”robot” (röd). Liksom fallet med två klasser segmenterar modellen alla robotar med tydliga konturer. Majoriteten av felklassificeringarna sker vid golvet och på väggarna. Detta beror troligtvis på att den träningsdata inte innehöll tillräckligt många olika typer av ”golv” och ”hinder”. Eftersom hinder kan se väldigt olika ut är det svårt att få modellen att känna igen alla olika typer av hinder. Därför är tanken att modellen ska klassa allt som inte ser ut som golv eller en robot ska klassas till ”hinder”. Anledningen till att den inte lyckas med detta är, precis som i modellen med två klasser, för att variationen på träningsdata är för låg.



Figur 6.22: Visar en segmenterad bild, från AB Volvos fabrik i Tuve, med tre klasser ”hinder” (blå), ”golv”(grön) och ”robot” (röd). Semantiska segmenterings-modellen har lyckats segmentera roboten i stortsett korrekt med endast små defekter. Den har dock klassificerat stora delar av golvet och väggarna felaktigt.

I figur 6.23 visas segmenteringen av en bild från simulationen som inte ingick i träningsdatan och där syns ett liknande resultat som i figur 6.22. Robotarna klassificeras korrekt medan modellen har problem med golv och hinder. Eftersom de uppvisar liknande felklassificeringar bekräftar det att den begränsade variation bidrar till förlusten av precision och att det inte enbart är domänskiftet som påverkar. Den bristande variationen gör det svårt att exakt avgöra hur modellen påverkas av domänskiftet från simuleringen till verkligheten. För att kunna göra en bättre utvär-

dering borde en ny modell tränas på mer varierad data. Även om det inte går att fast ställa exakt, tyder resultaten på att skillnaden mellan simulering och verkligheten inte är för stor att vara användbar. För att skapa ett kvantitativt resultat bör ett dataset från verkligheten annoteras och utvärderas på samma sätt som testdatasetet i avsnitt 6.3.2.



Figur 6.23: Figuren visar segmenteringen av en simulerad bild med en vinkel som inte ingick i träningsdatan. Segmenteringen har 3 klasser ”hinder” (blå), ”golv”(grön) och ”robot” (röd).Resultatet visar liknande felklassificeringar som av den verkliga bilden i figur 6.22.

Det positiva med resultaten som erhålls i detta avsnitt är att modellen alltid lyckas segmentera roboten. Det är den del av träningsdatan som haft mest fokus och därav förväntats visa högst precision. I alla de verkliga bilderna som har segmenterats har åtta av åtta robotar identifierats korrekt, med endast marginella fel i segmentering. Den höga precisionen av att segmentera ut robotarna visar att tillvägagångssättet för att skapa verklighetstrogen data med hjälp av 3DGS-miljöer i UE5 fungerar. Däremot visar de modellernas felklassificeringar av andra objekt att mer varierad data i form av fler vinklar och rörliga objekt krävs för en bättre utvärdering. Om ny träningsdata med bättre distribution genereras hade resultatet kunnat förbättras och det skulle bli enklare utvärdera hur kvalitén på splatten påverkar resultatet.

6.3.4 Möjliga Förbättringar

För vidare utveckling och utvärdering av datagenerering i UE5 rekommenderas att ny träningsdata skapas med större variation, i syfte att förbättra AI-modellens prestanda och möjliggöra en djupare utvärdering. För att få ett bättre resultat som enklare kan utvärderas. Variation kan ökas genom att använda fler kameraperspektiv och unika scenarion. För att få fler användbara perspektiv krävs en bättre 3DGS-modell som har hög visuell kvalitet i samtliga vinklar. Fler unika situationer kan simuleras genom att dynamiskt skapa och flytta runt objekt av klassen ”hinder”, men

även genom att använda ljusskillnaderna och störningarna som skapades i kapitel 4. Antalet robotar inuti simuleringen borde göras dynamiskt för att öka träningsdatans variation ytterligare. För att få fler verkliga scenarier kan robotarnas beteende ändras till något mer logiskt än bara röra sig till slumpmässiga positioner, och andra rörliga objekt som människor och gaffeltruckar bör inkluderas i simuleringen. Även AI-modellen som används kan förbättras genom att ändra till en större eller modernare modell, men framförallt genom att optimera träningen och hyperparametrarna.

Innan storskalig generering görs bör bildtagningen optimeras, eftersom det är en tydlig flaskhals i processen. Ett möjligt alternativ skulle vara att ta skärmdumpar av kamerornas vyer under simuleringens gång, istället för att använda Movie Render Queue. Genom att bygga om strukturen liknande ett multiplayer-spel, där varje kamera är en spelare, hade bildtagningen av alla kameror kunnat ske parallellt istället för sekventiellt. Vid användning av skärmdumpar renderas inte heller några överflödiga bilder av postprocessing-materialet, vilket effektiviserar processen.

6.4 Samhälleliga och etiska aspekter

Vid utvecklingen av en realistisk digital tvilling av en industriell miljö med hjälp av 3D Gaussian Splatting (3DGS), uppstår flera samhälleliga och etiska frågor kring användningen. Ur ett etiskt perspektiv är följande centrala delar viktiga att ta hänsyn till: att bidra till nytta, inte orsaka skada eller inskränka på andras autonomi och integritet. I detta avsnitt presenteras en analys av samhälleliga och etiska aspekter.

Implementeringen av en realistisk digital tvilling inom industrimiljöer kan automatiseringen effektiviseras, vilket i sin tur kan sänka produktionskostnaderna, öka produktiviteten och flexibiliteten samt främja hållbarhet [38]. Detta kan bidra till en positiv samhällspåverkan i form av högre effektivitet, minskad miljöpåverkan och förbättrade arbetsförhållanden.

Däremot finns det utmaningar och potentiella nackdelar ur etiska samt samhälleliga perspektiv. En ökad automatisering kan minska behovet av mänsklig arbetskraft, vilket kan leda till arbetsförluster [39]. Detta kan i sin tur påverka individer och samhällen negativt, särskilt inom yrkesgrupper där manuellt arbete utgör en central del av sysselsättningen.

En digital tvilling skapad med 3DGS är beroende av ett perceptionssystem och högkvalitativ data. Brister i data eller funktionalitet kan orsaka säkerhetsrisker i industrimiljöer, vilket kan resultera i allvarliga skador för arbetarna. Detta väcker etiska frågor kring systemets implementering och säkerhet. Det uppstår även dilemman kring datainsamling och integritet, eftersom träning av digitala tvillingar kräver omfattande insamling av data. Detta är även relevant för objektidentifiering vid implementering av autonoma system. Beroende på datakällor och insamlingsmetoder kan detta leda till integritetsintrång, särskilt om personliga eller känsliga uppgifter samlas in utan samtycke eller tillräckliga skyddsåtgärder [40].

Utvecklingen av en realistisk digital tvilling skapad med 3DGS inom industrimiljöer medför både möjligheter och utmaningar ur ett samhälleligt och etiskt perspektiv. För att säkerställa en säker implementering är det nödvändigt att noggrant analysera och hantera potentiella risker.

6.5 Framtida utveckling

En viktig utvecklingsriktning är att utöka storleken på de modellerade miljöerna. Detta projekt fokuserade på detaljerade modeller av begränsade ytor. Framtida studier bör undersöka möjligheten att modellera större områden, såsom kompletta fabrikslayouter, för att möjliggöra simulering av mer komplexa logistiska scenarier och träning av robotar för fullskalig autonom navigering. För att förbättra kvaliteten på dessa simuleringar är det även centralt att optimera överföringen av 3DGS-modeller till UE5. Den observerade kvalitetsförlusten vid denna process, se avsnitt 6.1.4, bör analyseras för att identifiera underliggande orsaker och utveckla strategier för att minimera den. Detta inkluderar även hantering av problem vid rendering av flera sammansatta 3DGS-modeller i UE5, se figur 6.12 i avsnitt 6.1.4.

En annan viktig aspekt är att möjliggöra mer avancerade dynamiska simuleringar. För att öka robustheten och generaliserbarheten hos tränade maskininlärningsmodeller är det nödvändigt att införa ytterligare dynamiska element i den digitala tvillingen. Utöver simulering av rörliga objekt bör detta inkludera mer varierade simuleringar av ljusförhållanden, se avsnitt 6.1.2, och potentiellt andra sensoriska störningar, vilket kan bidra till mer realistisk träningsdata för AI-modeller. För att öka variationen i träningsdatan (avsnitt 5.1), bör även simulering av ett större antal dynamiska objekt som inte är primärt intressanta, såsom slumpmässigt placerade pallar, gående människor och fordon som står i vägen, implementeras. Metoder för att automatiskt variera antalet karaktärer och deras positioner kan också utforskas för att generera en bredare uppsättning unika scenarier.

Vidare bör användningen av 360-kameror för datainsamling undersökas. Framtida forskning bör utvärdera fördelar och nackdelar med denna metod, såsom ökad täckning kontra potentiell kvalitetsförlust. Som tidigare nämnts i rapporten, se avsnitt 6.1.2, var det utmanande att erhålla fullständig bilddata med konventionella metoder, vilket resulterade i problem med rekonstruktionen av plana ytor och visuella artefakter.

För att öka effektiviteten i datagenereringen bör metoder för att snabba upp bildgenereringsprocessen undersökas. Den nuvarande användningen av Movie Render Queue utgör en flaskhals, se avsnitt 5.1. Alternativa metoder som att ta skärmdumpar direkt från kameravyerna under simulering eller att utforska en arkitektur liknande ett multiplayer spel där varje kamera representeras av en spelare, skulle kunna möjliggöra snabbare och potentiellt parallell bildtagning. Vidare bör optimeringen av den tränade AI-modellen fortsätta. Detta inkluderar att finjustera hyperparametrar som antalet träningsomgångar och inlärningshastighet, samt att eventuellt utforska mer avancerade eller större AI-modeller. Även alternativa träningsstrategier, som att hårdare vikta felklassificeringar av specifika klasser (t.ex. robotar och golv), kan utvärderas. Slutligen kan det även vara relevant att undersöka hur en objektigenkänningsalgoritm skulle prestera på den data som blev genererad.

För att kvantitativt kunna utvärdera kvalitet och realism hos den genererade data

ur ett maskininlärningsperspektiv, bör framtida arbete även inkludera framtagning och användning av ett manuellt annoterat dataset från de verkliga miljöerna (t.ex. Volvos fabrik). Att träna och testa modeller på både simulerad och verkligt annoterad data skulle ge värdefulla insikter i vilka aspekter av simuleringen som behöver förbättras för att den ska bli så likvärdig verklig data som möjligt. En sådan jämförelse skulle också möjliggöra en starkare validering av nyttan med att skapa ett automatiskt annoterat ground-truth-dataset med den 3DGS.

Realistiskt karaktärsbeteende är också ett viktigt utvecklingsområde. För att öka realismen i simuleringarna bör karaktärers beteende förbättras. Detta kan inkludera implementering av kollisionsundvikande, och mer naturliga rörelser vilket kräver utveckling/användning av mer avancerade system/modeller.

7
Slutsats

Användningen av 3D Gaussian Splatting (3DGS) för att skapa digitala tvillingar har visat på stor potential. Tekniken möjliggör konstruerandet av digitala representationer av verkliga miljöer, som visat en hög grad av överensstämmelse med verkligheten. Den genererade ground-truth-data har visat sig vara tillräckligt realistisk för att möjliggöra träning av en semantisk segmenteringsmodell på enbart syntetisk data, som därefter kunnat tillämpas på verkliga bilder med goda resultat. Däremot har skapandet av 3DGS-modeller visat på utmaningar kring bildtagning av miljöer, vilket påverkar rekonstruktionen negativt. Samtidigt försämras kvaliteten av 3DGS-modeller vid importering till UE5. Sammantaget har dessa faktorer försvårat processen att skapa ett varierat och verklighetstroget dataset.

I dagsläget producerar Postshot mycket detaljrika och verklighetstrogna 3DGS-modeller. Om implementeringen av 3DGS i simuleringsprogrammen utvecklas till samma nivå, kan detta i hög grad förbättra möjligheterna att simulera verkliga miljöer samt generera ground-truth-data. Eftersom datagenerering i simulerade miljöer är avsevärt enklare och mindre tidskrävande än manuell annotering, kan detta i förlängningen underlätta träning av allt mer komplexa maskininlärningsmodeller.

Källförteckning

[1] Infosys BPM, "Rise of the machines: Robotics' impact on the evolution of manufacturing," Hämtad Feb. 13, 2025. [Online]. Tillgänglig: https://www.infosysbpm.com/blogs/manufacturing/robotics-in-manufacturing.html.

[2] Volvo Group, "Innovation in motion! Introducing a few of our latest colleagues - automated transporters powered by AI and computer vision," Linkedin.com, Hämtad Feb. 12, 2025. [Online]. Tillgänglig: https://www.linkedin.com/posts/volvo-group_follow-us-for-more-stories-volvo-group-activity-7259523397385117697-ikCa.

[3] IBM, "What is a digital twin?" Hämtad Feb. 14, 2025, [Online]. Tillgänglig: https://www.ibm.com/think/topics/what-is-a-digital-twin.

[4] F. Tao, H. Zhang, A. Liu och A. Y. C. Nee, "Digital Twin in Industry: State-of-the-Art," *IEEE Transactions on Industrial Informatics*, årg. 15, nr 4, s. 2405–2415, 2019. doi: 10.1109/TII.2018.2873186.

[5] F. Tao och M. Zhang, "Digital Twin Shop-Floor: A New Shop-Floor Paradigm Towards Smart Manufacturing," *IEEE Access*, årg. 5, s. 20 418–20 427, 2017. doi: 10.1109/ACCESS.2017.2756069.

[6] F. Bernardini och H. Rushmeier, "The 3D model acquisition pipeline," *Computer Graphics Forum*, årg. 21, 2 2002, ISSN: 01677055. doi: 10.1111/1467-8659.00574.

[7] C. K. Chua, C. H. Wong och W. Y. Yeong, "Software and Data Format," *Standards, Quality Control, and Measurement Sciences in 3D Printing and Additive Manufacturing*, s. 75–94, jan. 2017. doi: 10.1016/B978-0-12-813489-4.00004-0. https://www.sciencedirect.com/science/article/pii/B9780128134894000040#s0115.

[8] J. Cummings, "File:Monmouth castle point cloud, created with Photosynth 02.jpg," commons.wikimedia.com, Hämtad: Apr. 29, 2025. [Online]. Tillgänglig: https://commons.wikimedia.org/wiki/File:Monmouth_castle_point_cloud,_created_with_Photosynth_02.jpg.

[9] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey och J. M. Reynolds, "'Structure-from-Motion' photogrammetry: A low-cost, effective tool for geoscience applications," *Geomorphology*, årg. 179, s. 300–314, dec. 2012, ISSN: 0169-555X. doi: 10.1016/J.GEOMORPH.2012.08.021. https://www.sciencedirect.com/science/article/pii/S0169555X12004217.

[10] Wikipedia, "File:Sfm.jpg," commons.wikimedia.org, Hämtad: Maj. 12, 2025. [Online]. Tillgänglig: https://commons.wikimedia.org/wiki/File:Sfm.jpg.

[11] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi och R. Ng, "NeRF: representing scenes as neural radiance fields for view synt-

hesis," *Commun. ACM*, årg. 65, s. 99–106, 1 dec. 2021, ISSN: 0001-0782. doi: 10.1145/3503250. https://doi.org/10.1145/3503250.

[12] B. Kerbl, G. Kopanas, T. Leimkuehler och G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Transactions on Graphics*, årg. 42, 4 2023, ISSN: 15577368. doi: 10.1145/3592433.

[13] A. Basso, F. Condorelli, A. Giordano, S. Morena och M. Perticarini, "EVOLUTION OF RENDERING BASED ON RADIANCE FIELDS. THE PALERMO CASE STUDY FOR A COMPARISON BETWEEN NERF AND GAUSSIAN SPLATTING," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, årg. XLVIII-2/W4-2024, s. 57–64, 2024. doi: 10.5194/isprs-archives-XLVIII-2-W4-2024-57-2024. https://isprs-archives.copernicus.org/articles/XLVIII-2-W4-2024/57/2024/.

[14] Z. Wang, A. Bovik, H. Sheikh och E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, årg. 13, nr 4, s. 600–612, 2004. doi: 10.1109/TIP.2003.819861.

[15] Jawset, "Postshot User Guide," Hämtad: Apr. 30, 2025. [Online]. Tillgänglig: https://www.jawset.com/docs/d/Postshot+User+Guide.

[16] Jawset, "Training Configuration," Hämtad: Apr. 30, 2025. [Online]. Tillgänglig: https://www.jawset.com/docs/d/Postshot+User+Guide/Interface/Training+Configuration.

[17] O. Huttunen, "Unlocking the Mystery of Sparse Point Clouds in Gaussian Splatting," YouTube, Aug. 15, 2024, [Video]. Tillgänglig: https://www.youtube.com/watch?v=RoZg_-Npi0E&t=309s&ab_channel=OlliHuttunen.

[18] Jawset, "Viewport Tools," Hämtad: Apr. 30, 2025. [Online]. Tillgänglig: https://www.jawset.com/docs/d/Postshot+User+Guide/Interface/Viewport+Tools.

[19] O. Huttunen, "Cleaning Gaussian Splatting models in Postshot," YouTube, Aug. 25, 2024, [Video]. Tillgänglig: https://www.youtube.com/watch?v=2CoojdyKFXc&t=83s&ab_channel=OlliHuttunen.

[20] Jawset, "Getting Started," Hämtad: Apr. 30, 2025. [Online]. Tillgänglig: https://www.jawset.com/docs/d/Postshot+User+Guide/Getting+Started.

[21] A. Regalbuto, "How we wrote a GPU-based Gaussian Splats viewer in Unreal with Niagara," magnopus.com, Hämtad: 12 maj 2025. [Online]. Tillgänglig: https://www.magnopus.com/blog/how-we-wrote-a-gpu-based-gaussian-splats-viewer-in-unreal-with-niagara.

[22] Epic Games, "Unreal Engine," unrealengine.com, Hämtad Feb. 12, 2025, [Online]. Tillgänglig: https://www.unrealengine.com/en-US.

[23] DANTHREE STUDIO, "What Is A 3D Mesh Model? (Definition Examples)," Hämtad Mar. 25, 2025. [Online]. Tillgänglig: https://www.danthree.studio/en/blog-cgi/what-is-a-3d-mesh-model-definition-examples.

[24] Epic Games, "Working with Plugins," unrealengine.com, Hämtad Feb. 12, 2025. [Online]. Tillgänglig: https://dev.epicgames.com/documentation/en-us/unreal-engine/working-with-plugins-in-unreal-engine.

[25] F. Tao, M. Zhang och A. Nee, *Digital Twin Driven Smart Manufacturing*. Academic Press, 2019, ISBN: 978-0-12-817630-6.

[26] Epic Games, “Directional Lights in Unreal Engine,” Hämtad: Maj. 8, 2025. [Online]. Tillgänglig: https://dev.epicgames.com/documentation/en-us/unreal-engine/directional-lights-in-unreal-engine.

[27] Epic Games, “Dirt Mask – Post Process Effects in Unreal Engine,” Hämtad: Maj. 5, 2025. [Online]. Tillgänglig: https://dev.epicgames.com/documentation/en-us/unreal-engine/post-process-effects-in-unreal-engine#dirtmask.

[28] A. Arts, “Lens Dust - Free texture pack.” https://courses.azielarts.com/register-lensdust-download.

[29] IBM, “What is semantic segmentation?” Hämtad Feb. 12, 2025. [Online]. Tillgänglig: https://www.ibm.com/think/topics/semantic-segmentation.

[30] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy och A. L. Yuille, “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,” 2017. arXiv: 1606.00915 [cs.CV]. https://arxiv.org/abs/1606.00915.

[31] T. Krantz och A. Jonker, “What is ground truth?” ibm.com, Hämtad Feb. 13, 2025 [Online]. Tillgänglig: https://www.ibm.com/think/topics/ground-truth.

[32] G. Csurka, R. Volpi och B. Chidlovskii, “Semantic Image Segmentation: Two Decades of Research,” 2023. arXiv: 2302.06378 [cs.CV]. https://arxiv.org/abs/2302.06378.

[33] Pytorch team, “Deeplabv3,” pytorch.org, Hämtad: 14 april 2025. [Online]. Tillgänglig: https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/.

[34] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn och A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision*, årg. 88, nr 2, s. 303–338, juni 2010.

[35] T.-Y. Lin, M. Maire, S. Belongie m. fl., “Microsoft COCO: Common Objects in Context,” 2015. arXiv: 1405.0312 [cs.CV]. https://arxiv.org/abs/1405.0312.

[36] R. Azad, M. Heidary, K. Yilmaz m. fl., “Loss Functions in the Era of Semantic Segmentation: A Survey and Outlook,” 2023. arXiv: 2312.05391 [cs.CV]. https://arxiv.org/abs/2312.05391.

[37] A. Mortazi, V. Cicek, E. Keles och U. Bagci, “Selecting the Best Optimizers for Deep Learning based Medical Image Segmentation,” 2023. arXiv: 2302.02289 [eess.IV]. https://arxiv.org/abs/2302.02289.

[38] S. Ma, W. Ding, Y. Liu, S. Ren och H. Yang, “Digital twin and big data-driven sustainable smart manufacturing based on information management systems for energy-intensive industries,” *Applied Energy*, årg. 326, s. 119 986, 2022, ISSN: 0306-2619. doi: https://doi.org/10.1016/j.apenergy.2022.119986. https://www.sciencedirect.com/science/article/pii/S0306261922012430.

[39] GAO, “Which Workers Are the Most Affected by Automation and What Could Help Them Get New Jobs?” Hämtad Feb. 14, 2025. [Online]. Tillgänglig: https://www.gao.gov/blog/which-workers-are-most-affected-automation-and-what-could-help-them-get-new-jobs.

[40] GAO, “Science Tech Spotlight: Digital Twins—Virtual Models of People and Objects,” Hämtad: Maj. 9, 2025. [Online]. Tillgänglig: https://www.gao.gov/products/gao-23-106453?utm_campaign=usgao_email&utm_content=daybook&utm_medium=email&utm_source=govdelivery.

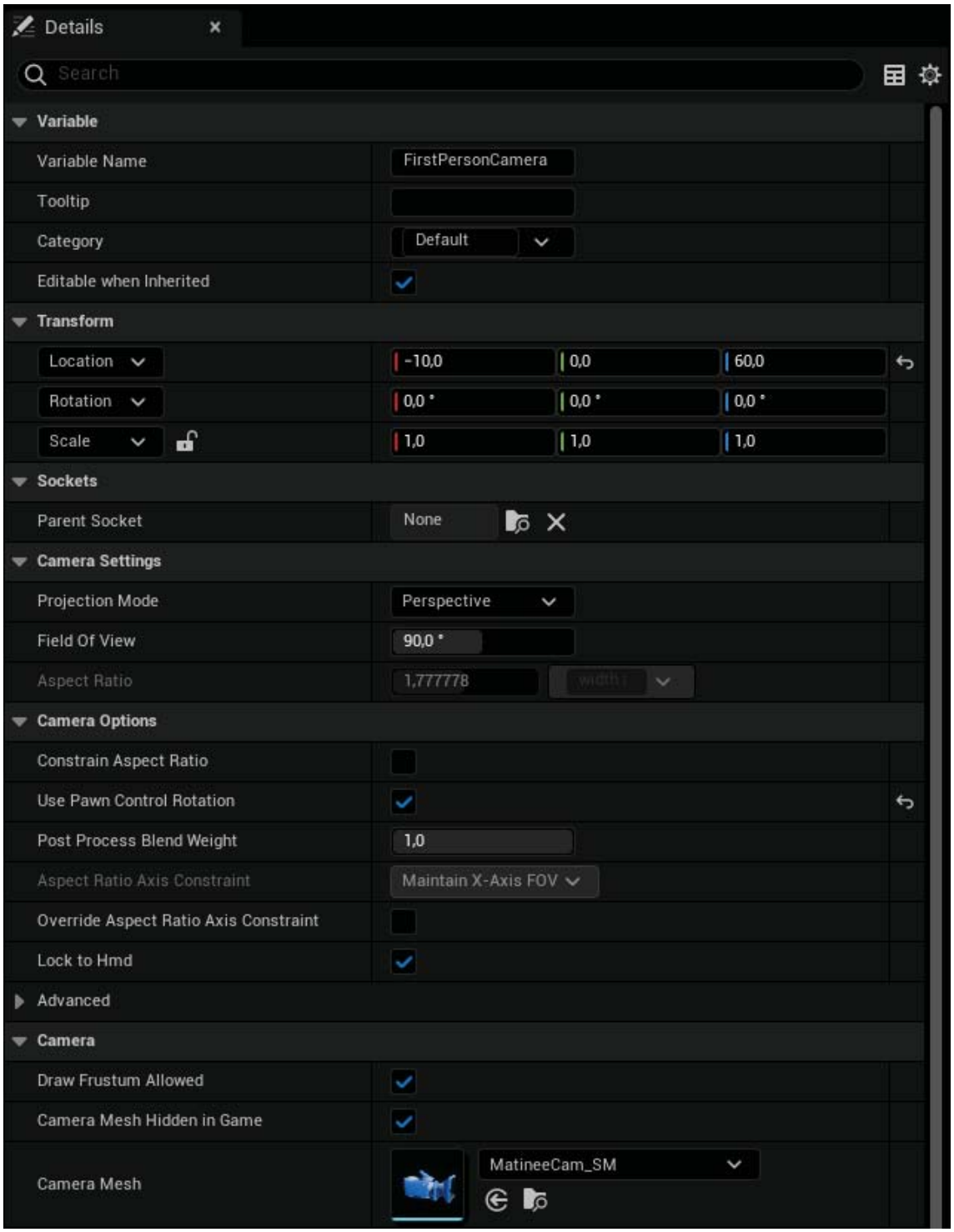
A

Appendix 1

A.1 Tuvefabriken

Figur A.1: Figuren illustrerar genomförandet av drönarflygningen.

A.2 UE5 details panel



Figur A.2: Details-fönster för ett kameraobjekt i Unreal engine 5.

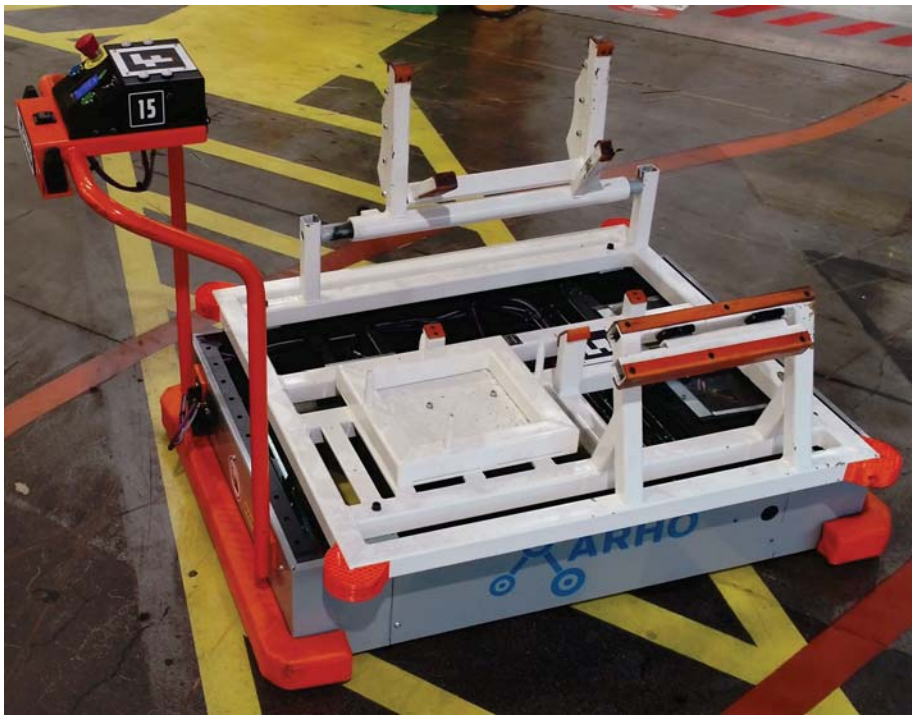
A.3 Gaffeltruck

Figur A.3: Foto av en verklig gaffeltruck.

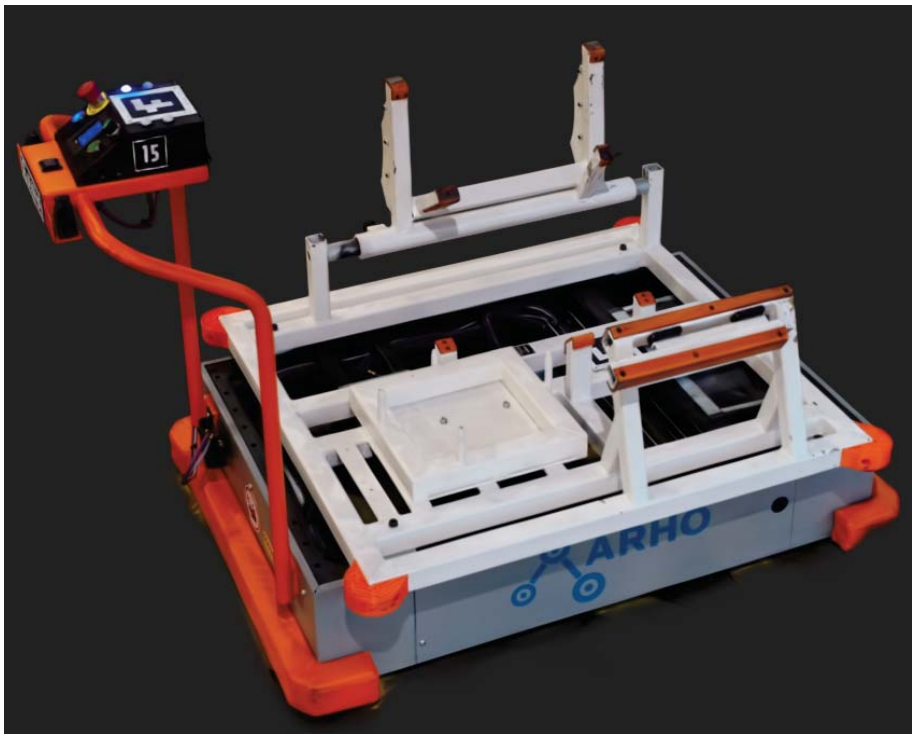
Figur A.4: Bild på en gaffeltruck i Postshot.

Figur A.5: Bild på en gaffeltruck importerad i UE5.

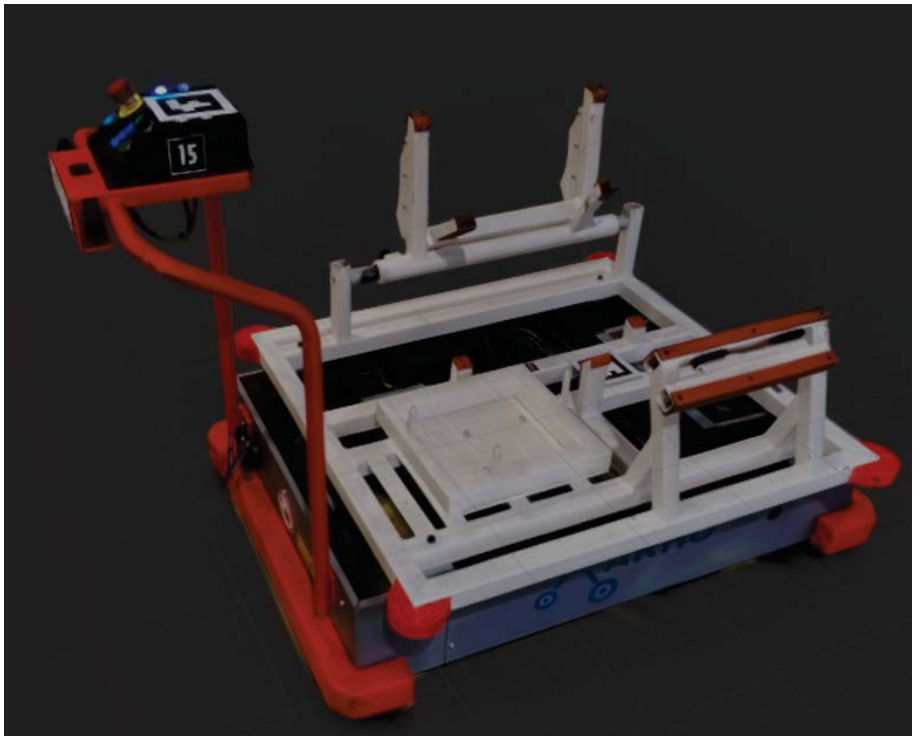
A.4 Robot utan last



Figur A.6: Foto av en verklig robot utan last.

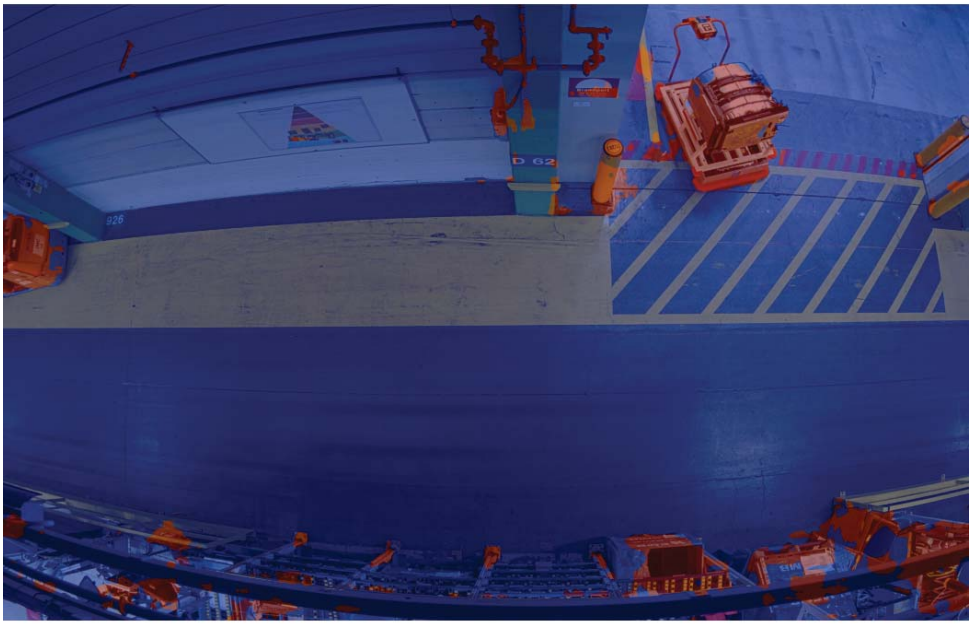


Figur A.7: Bild på en robot utan last i Postshot.

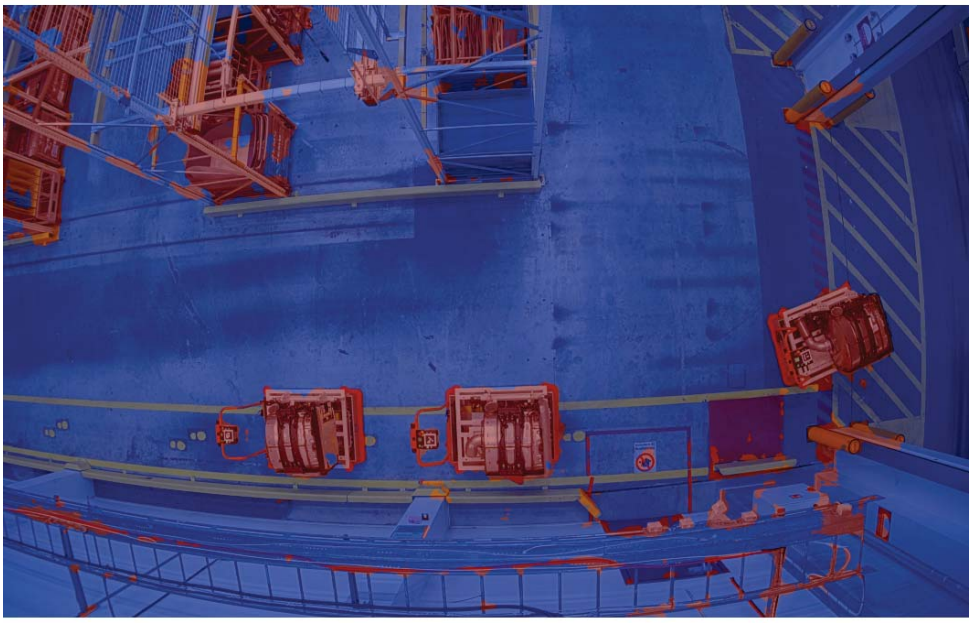


Figur A.8: Bild på en robot utan last importerad i UE5.

A.5 AI-segmentering med två klasser

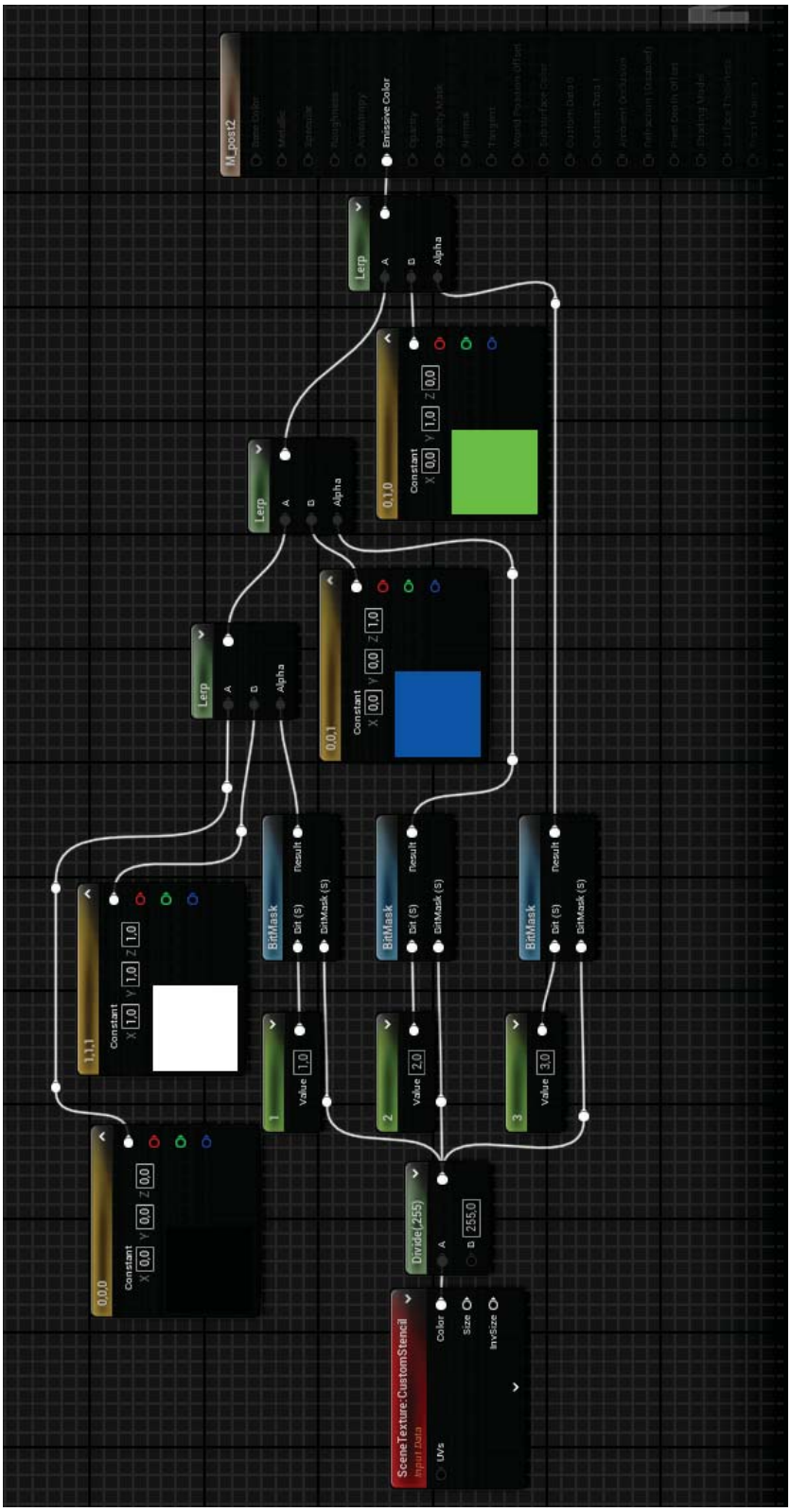


Figur A.9: Två klasser: klassificerad bild från AB Volvos fabrik i Tuve. Innehåll: en robot och bakdelen av en gaffeltruck (till vänster i bild).



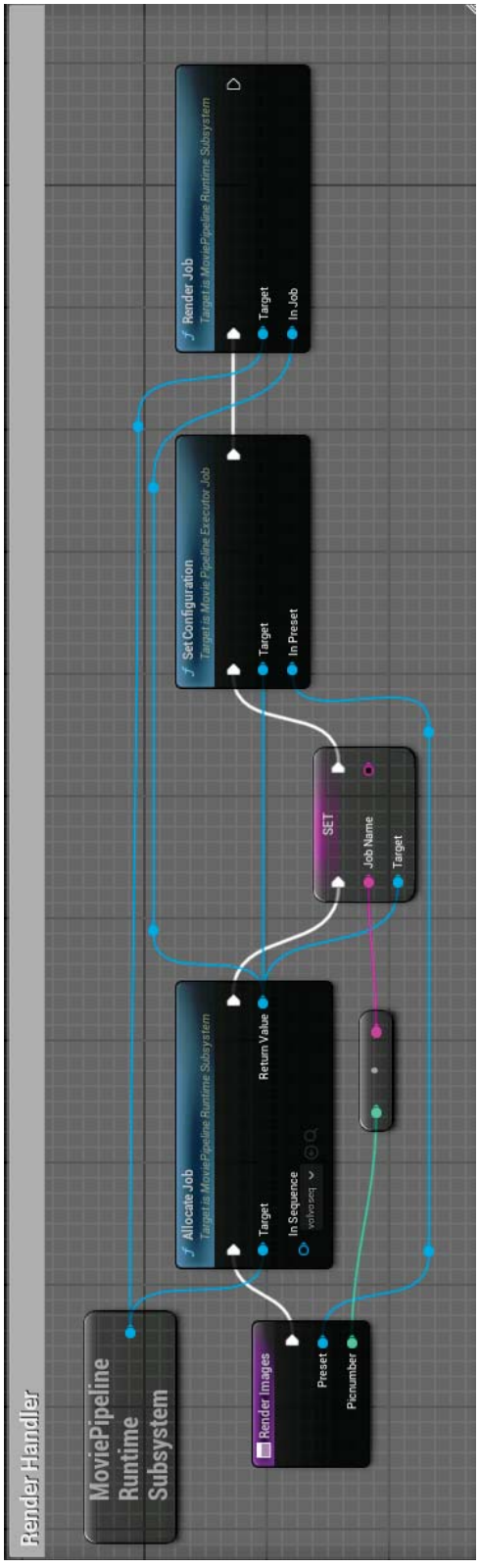
Figur A.10: Två klasser: klassificerad bild från AB Volvos fabrik i Tuve. Innehåll: tre robotar och förvaringshyllor.

A.6 Segmenterings postprocessing material



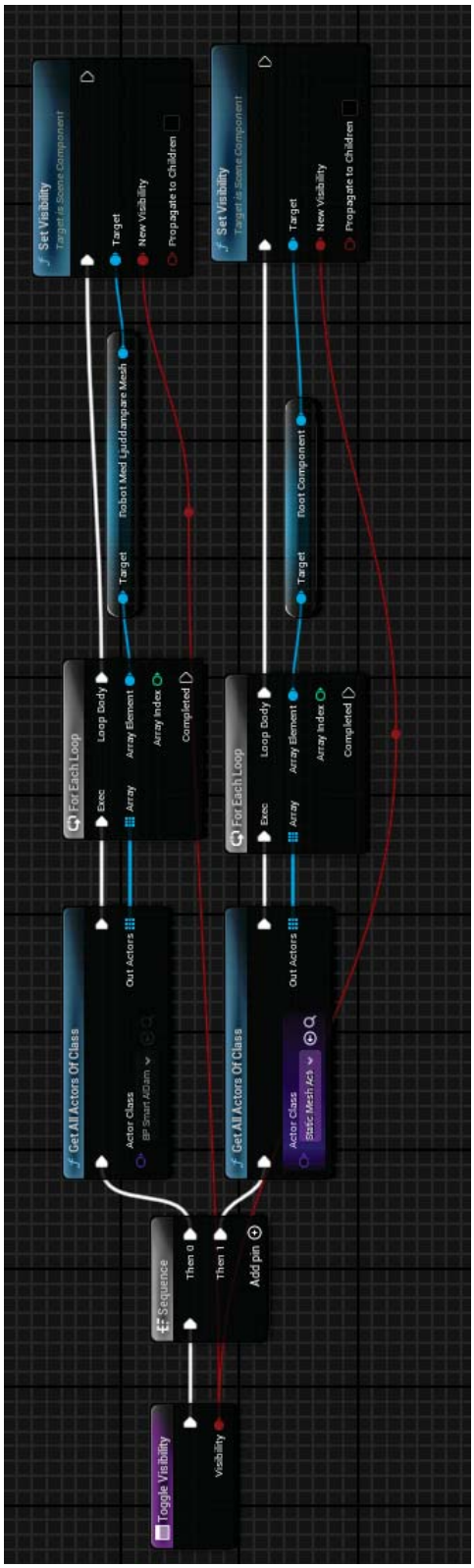
Figur A.11: Blueprint-funktion för att rendera bilder.

A.7 Renderingsfunktion



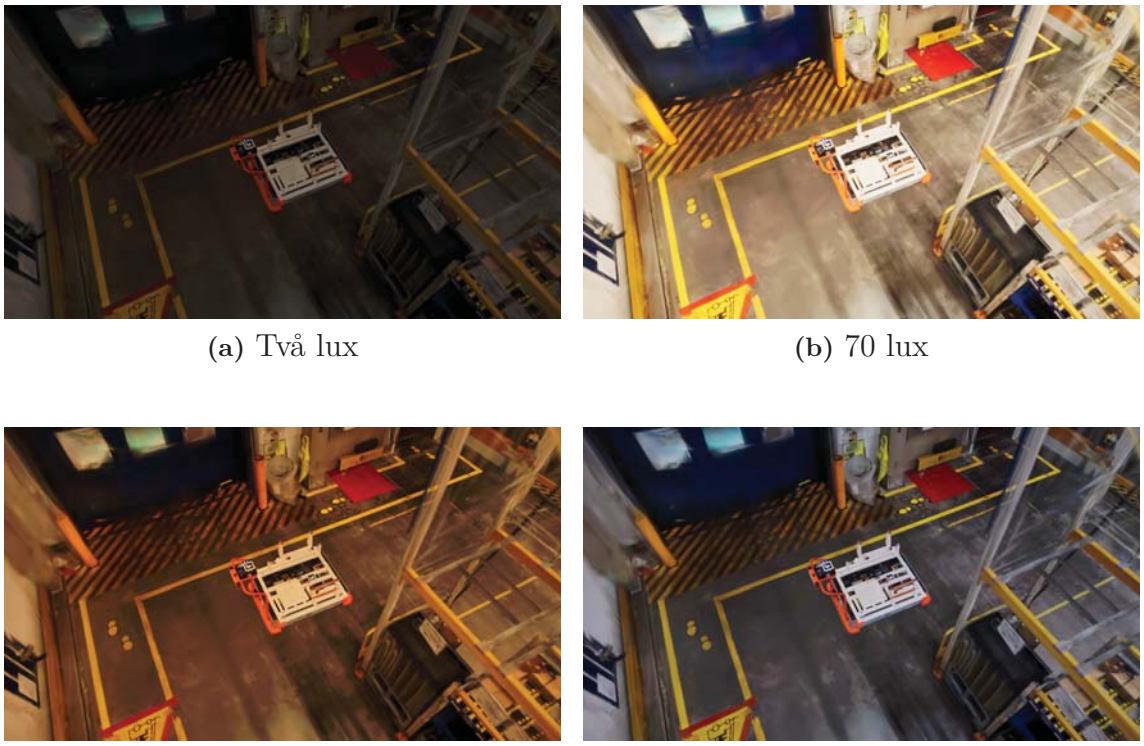
Figur A.12: Blueprint-funktion för att rendera bilder.

A.8 Växla synlighetsfunktion



Figur A.13: Blueprint funktion som växlar synlighet hos samtliga meshar.

A.9 Simulerade ljusskillnader och störningar



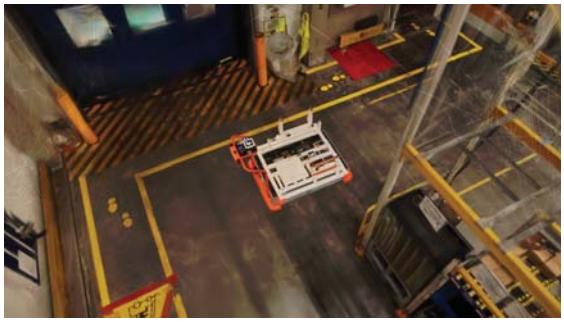
(a) Två lux

(b) 70 lux

(c) Temperatur 4000

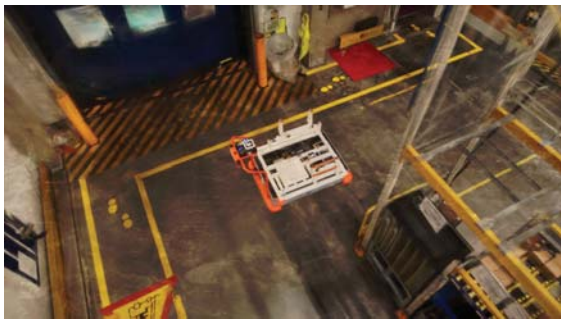
(d) Temperatur 16000

Figur A.14: Figurerna illustrerar simulerade ljusskillnader gjorda med hjälp av Directional Light i UE5. Exemplen innehåller låg respektive hög ljusstyrka, samt varmare och kallare ljus.



(a) Spindelväv framför kameralinsen.

(b) Grovkornigt damm på kameralinsen.



(c) Fingeravtryck på kameralinsen.

(d) Finkornit damm på kameralinsen.

Figur A.15: Figurerna visar diverse simulerade störningar med hjälp av Dirt Masks i UE5.

A.10 Datorspecifikation

- AMD Ryzen 9 9950X
- RTX 2080Ti
- 256GB DDR5 5600Mhz
- NVME 4TB SSD