CHALMERS

Fotorealistisk simulering av mobila robotar i fabriksmiljöer

Utvärdering av 3DGS-baserade simuleringsmiljöer för träning av segmenteringsmodeller och styrning av mobila robotar

JONATHAN BJÖRKMAN , SIGRID EKSTRÖM, FILIP HÅKANSSON, VALERIJA KAZLAUSKAITE, ADAM SUNNAR, HUGO WALLIN

Institutionen för Elektroteknik

CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2026
www.chalmers.se

Kandidatarbete 2026

Fotorealistisk simulering av mobila robotar i en fabriksmiljö

Design av en fotorealistisk simulering av mobila robotar i en fabriksmiljö

JONATHAN EDVINSSON BJÖRKMAN
SIGRID EKSTRÖM
FILIP HÅKANSSON
VALERIJA KAZLAUSKAITE
ADAM SUNNAR
HUGO WALLIN

CHALMERS

Institutionen för Elektroteknik
Chalmers Tekniska Högskola
Göteborg, Sverige 2026

Fotorealistisk simulering av mobila robotar i en fabriksmiljö
Design av en fotorealistisk simulering av mobila robotar i en fabriksmiljö

JONATHAN EDVINSSON BJÖRKMAN , SIGRID EKSTRÖM, FILIP HÅKANSSON, VALERIJA KAZLAUSKAITE, ADAM SUNNAR, HUGO WALLIN

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

Kandidatarbete 2026
Institutionen för Elektroteknik
EENX16-26-16
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Omslagsbild: 3D Gaussian splatt-modell av Volvos fabrik i Tuve.

Skriven i LaTeX
Gothenburg, Sweden 2026

Abstract

Modern factories handle more components than ever before, creating major logistical challenges. As a solution, autonomous mobile robots (AMRs) are now being used. Training the behavior of these robots in real-world environments is expensive and time-consuming. For this reason, the possibility of using simulated environments built with 3D Gaussian Splatting (3DGS) is being investigated as an alternative to real-world training.

This bachelor's thesis evaluates whether the simulation environment can be used to collect synthetic data for a segmentation model for object localization, which in turn can enable the control of AMRs. The method involved filming industrial environments which could then be used to create a digital twin using a 3DGS model. From the 3D model, a simulation environment was created through synthetic dynamic and static objects that replicate the real industrial environment. Synthetic data was then collected from the simulation environment which could be used to train a segmentation model. The fine-tuned model could be used for object localization and identification, which was utilized for the control of AMRs.

The results showed that it is possible to produce photorealistic simulation environments of industrial environments, from which it is possible to produce large quantities of high-quality annotated data automatically. The instance segmentation model YOLOv8-Seg Nano was able to localize objects in the simulation environment in real time. Consequently, AMRs could be controlled using the segmented instances.

The conclusion is that although perfect photorealism was not achived in this project, 3DGS proves to be a viable tool in towards complete realism. The resulting environment is however good enough to create synthetic training data that results in a segmentation model capable of predicting objects in real world photos.

Keywords: 3D Gaussian Splatting, LiDAR, Real-Time Simulation, Segmentation, Machine Learning, Digital Twin.

Sammandrag

Moderna fabriker hanterar fler komponenter än någonsin tidigare, vilket skapar stora logistiska utmaningar. Som lösning finns idag autonoma mobila robotars (AMR). Att träna dessa robotars beteende i verkligheten är dyrt och tidskrävande. Av denna anledning undersöks möjligheten att använda simuleringsmiljöer byggda med 3D Gaussian Splatting (3DGS) som alternativ för verklig träning.

Detta kandidatarbete utvärderar om simuleringsmiljön kan användas till insamling av syntetisk data för en segmenteringsmodell vars uppgift är objektlokalisering. I sin tur undersöks möjligheten att styra virtuella AMR:er med hjälp av modellen.

Metoden innefattade filmning av industrimiljöer som sedan kunde användas till att skapa en digital tvilling med hjälp av en 3DGS. Utifrån 3DGS modellen skapades en simuleringsmiljö bestående av dynamiska och statiska objekt som förekommer i den verkliga industrimiljön. Därefter skapades syntetisk data i simuleringsmiljön som användes till träning av en segmenteringsmodell. Den finjusterade modellen kunde användas till objektlokalisering samt identifiering som nyttjades till styrning av AMR:er.

Resultatet visade att det är möjligt att framställa nästintill fotorealistiska simuleringsmiljöer av industrimiljöer varifrån det är möjligt att producera stora mängder annoterad data automatiskt. Instanssegmenteringsmodellen YOLOv8-Seg Nano kunde lokalisera objekt i simuleringsmiljön i realtid. Följaktligen kunde AMR:er styras med hjälp av de segmenterade instanserna.

Slutsatsen är att även om perfekt fotorealism inte uppnåddes i detta projekt, visar 3DGS sig vara ett användbart verktyg på vägen mot fullständig realism. Den resulterande miljön är däremot tillräckligt bra för att skapa syntetisk träningsdata som leder till en segmenteringsmodell med förmågan att identifiera objekt i verkliga bilder.

Nyckelord: 3D Gaussian Splatting, LiDAR, Realtidssimulering, Segmentering, Maskininlärning, Digital Tvilling.

Tillkännagivande

Vi vill rikta ett stort tack till professor Knut Åkesson för hans handledning och stöd genom hela arbetets gång. Vi vill även tacka Martin Fabian för hans insats som examinator samt Per Nyqvist för värdefullt teknisk stöd. Slutligen vill vi rikta ett stort tack till AB Volvo samt Atieh Hanna och Kristofer Bengtsson för deras värdefulla stöd och för att ha varit vår kontakt med företaget under arbetets gång.

AI-användning

I detta arbete har AI använts för att ge återkoppling på språk, struktur och upplägg av rapporten.

Jonathan Björkman Edvinsson, Sigrid Ekström, Filip Håkansson, Valerija Kazlauskaite, Adam Sunnar, Hugo Wallin, Gothenburg, Maj 2026

Akronymer

Följande lista innehåller akronymer som används i detta examensarbete, listade i alfabetisk ordning:

3DGS	3D Gaussian Splatting
AMR	Automated Mobile Robot
CNN	Convolutional Neural Network
FPS	Frames Per Second
IoU	Intersection Over Union
LCC	Lixel CyberColor
LiDAR	Light Detection and Ranging
MLP	Multilayer Perceptron
MiT	Mixed Transformer
ROS 2	Robot Operating System 2
RGB	Red Green Blue
UE5	Unreal Engine 5
YARP	Yet Another Robot Platform

Innehåll

Figurer

Tabeller

1
Inledning

Moderna fabriker hanterar fler komponenter än någonsin tidigare. Detta skapar stora logistiska utmaningar som fortsatt förväntas växa. En särskilt utsatt industri är bil- och fordonsindustrin där det sker en storskalig omvandling till elektriska fordon samtidigt som fordon med traditionell drift fortfarande tillverkas [1]. För att hantera detta växande problem utvecklas nya autonoma lösningar på flera håll. Det här projektet undersöker därav möjligheten att kunna simulera de komplexa autonoma systemen som finns i dagens fabriksmiljöer med hjälp av digitala tvillingar.

1.1 Bakgrund

På AB Volvo sker utvecklingen i form av Automated Mobile Robots[1] (AMR) som kan utföra uppgifter inom logistik som tidigare utförts av människor. En av dessa uppgifter är transport av komponenter från lager till produktionslinje inom fabriken [2]. I fabriken tillverkas olika modeller på samma produktionslinje vilket kräver att rätt delar kommer till rätt plats vid rätt tillfälle för att bibehålla produktionshastigheten. Detta blir alltjämt svårare när antalet delar som hanteras ständigt ökar. För att möjliggöra användningen av autonoma robotar inom fabriken krävs möjlighet till igenkänning och klassificering av objekt såsom människor och hinder [1], även kallad perception. Detta kräver träning, vilket i nuläget oftast görs med manuellt skapade och annoterade träningsdata tagna från verkligheten.

För att skapa en väl fungerande modell krävs ett stort antal bilder med varierande innehåll [1]. Insamlingen och annotering[2] av data är en resurskrävande process som stör ordinarie verksamhet i lokalerna. Traditionella metoder för datainsamling medför även kraftiga utmaningar att skapa farliga, komplexa eller ovanliga scenarier så som brand, översvämning och udda ljusförhållanden. För att effektivisera denna process undersöks nu nya metoder för att träna modellerna.

Ett tidigare kandidatarbete i samarbete med AB Volvo i Tuve har visat hur det

[1] AMR skiljer sig från andra industrirobotar genom att vara centralt styrda. Sensorerna, i form av kameror, befinner sig inte på robotarna utan i taket och låter styralgoritmen se alla robotars position.

[2] Annotering syftar till att manuellt markera vad olika objekt är i en bild och fungerar som facit vid träning av segmenteringsmodell.

går att använda tekniken 3D Gaussian Splatting (3DGS) för att bygga upp digitala miljöer som sedan kan användas för träning av semantiska segmenteringsmodeller[3]. Metoden anses vara ”en lovande teknik för att effektivisera framtagning av träningsdata” [3, s. vi] och innebär då ett alternativ till resurskrävande manuella metoder. Däremot identifierade projektet problem med kvaliteten på den digitala miljön på grund av kvaliteten hos kameran som användes och mjukvaran som nyttjades [3].

Nästa steg i utvecklingen är att simulera robotarnas beteende, såsom styrning, i den digitala miljön. Där kan AMR-systemets utformning och dess prestanda bekräftas innan det implementeras i den verkliga fabriken. På så sätt kan AMR:er tillämpas i andra fabriker snabbare, effektivare och billigare [1].

1.2 Syfte

Syftet med detta projekt är att utveckla och utvärdera ett koncept för ett återkopplat system i form av en dynamisk realtidsbaserad[4] simuleringsmiljö av AB Volvos fabrik i Tuve. Detta utförs med hjälp av 3DGS för att skapa verklighetstrogna digitala tvillingar av industrimiljön. Konceptet ska öppna upp möjligheten att snabbt träna och validera autonoma system i nya miljöer.

1.3 Avgränsningar

Detta projekt avser att skapa en dynamisk simuleringsmiljö av en begränsad sektion av AB Volvos fabrik i Tuve. Projektet behandlar inte avancerad reglering av AMR:en, endast grundläggande reglering. Programmeringen av AMR:ens beteende avgränsas därför till att den ska stanna vid syn av hinder. Om ett mer avancerad beteende hos roboten önskas kan färdig utvecklade styralgoritmer att implementeras. Vid konstruktion av den dynamiska simuleringsmiljön kan sensorer och annan datainsamling komma att förenklas relativt till hur data samlas in i verkligheten. Att återskapa sensorer som odometrar och stötgivare virtuellt ligger utanför projektets omfång. Arbetet är avgränsat till att nyttja spelmotorn Unreal Engine 5 för skapandet av simuleringsmiljön. Arbetet kommer inte heller behandla andra metoder för att visualisera fabriken än 3DGS då en del av arbetets syfte är att utvärdera formatet.

Till sist ligger även en undersökning angående optimering av träningen och hyperparametrar av den valda semantiska segmenteringsmodellen utanför projektets begränsningar.

[3]En segmenteringsmodell är den autonoma identifieringen av innehållet i en bild eller video
[4]Realtid syftar på hur ofta simuleringen uppdateras.

1.4 Problem

För att kunna uppnå arbetets övergripande syfte kan projektet delas upp i följande tre delproblem: framtagning av en simuleringsmiljö, lokalisering av objekt och styrning av objekt. För att utvärdera dessa delproblem kommer följande frågeställningar att bearbetas samt besvaras.

- Vad är möjligheterna och begränsningarna med att använda 3D Gaussian Splatting för att skapa en realistisk simuleringsmiljö?
- Vilka är för och nackdelarna med att använda syntetisk träningsdata för lokalisering av simuleringsobjekt?
- Vad är svagheter och styrkor med att använda perception för att styra mobila robotar?

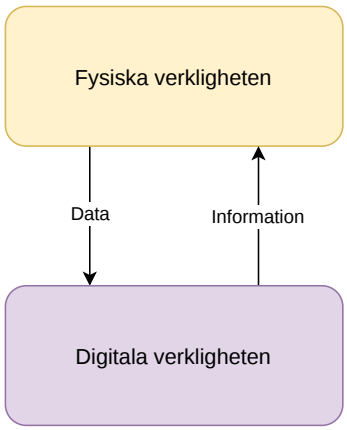
2
Digitala tvillingar

Detta kapitel avser att beskriva arbetet i sin helhet. Först kommer konceptet digitala tvillingar, som arbetet har sin grund i, att introduceras. Därefter kommer en övergripande arbetsprocess för varje identifierat delproblem att presenteras för att sedan sammanfattas.

2.1 Introduktion till digitala tvillingar

Den första konceptuella modellen för digitala tvillingar presenterades först år 2002 av Michael Grieves [4]. Vid den här tidpunkten benämndes dock inte konceptet som digitala tvillingar, utan presenterades som ett ideal för produktlivscykelhantering. Huvudsakligen bestod modellen av tre komponenter: den fysiska verkligheten, den digitala verkligheten samt kopplingen mellan dessa verkligheter. Det var inte förrän år 2010 som konceptet blev myntat digitala tvillingar av Grieves kollega på NASA [4]. Den generella idén bakom konceptet var fortfarande den samma: Att fysiska objekt bör kunna representeras som digitala objekt. Denna iden möjliggör i sin tur att arbete som traditionellt sett skett i den fysiska verkligheten nu kan ske i den digitala verkligheten.

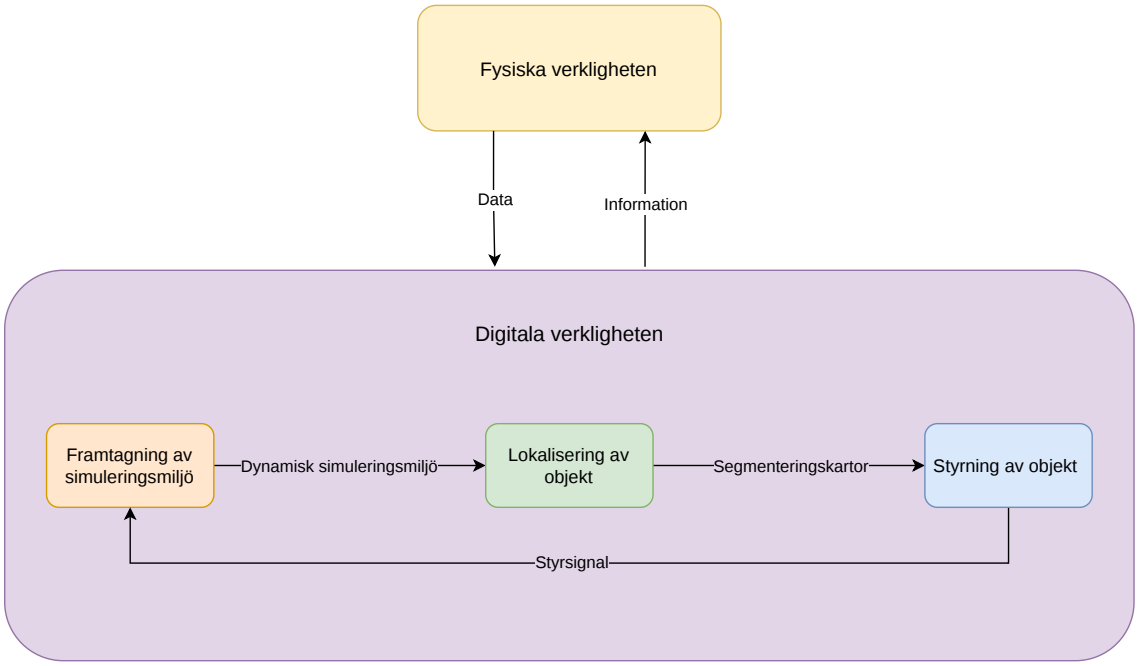
Enligt [5] finns ingen tydlig allmän överenskommelse om vad dagens modell för digitala tvillingar är. Därför avser detta arbete begreppet digitala tvillingar som den ovan beskrivna tredimensionella modellen, vilken illustreras nedanför.



Figur 2.1: Modell av digitala tvillingar med dimensionerna fysisk verklighet, digital verklighet och digital tråd.

Längst upp i Figur 2.1 visas den första komponenten i modellen för digitala tvillingar. Denna komponent är den fysiska verkligheten, vilken kan bestå av exempelvis fysiska system, objekt och/eller hela miljön vars entiteter består av. Längst ner visas den andra komponenten i modellen vilket är den digitala verkligheten. Denna komponent är en digital representation av den fysiska verkligheten, som innehåller informationen om samtliga entiteter i den fysiska verkligheten. Mellan dessa verkligheter visas den tredje komponenten vilket är kopplingen, även kallad för den digitala tråden [4]. Den digitala tråd ska representera informationsflödet mellan de båda verkligheterna, där realtidsinformation som beskriver entiteterna i den fysiska verkligheten fungerar som indata för den digitala verkligheten. Denna indata kan sedan användas för att uppdatera informationen som beskriver motsvarande digitaliserade entiteter inuti den digitala verkligheten. Med de uppdaterade digitaliserade entiteterna kan sedan den digitala tvillingen användas för att simulera den fysiska verkligheten.

Enligt [4] är den första fasen under implementeringen av digitala tvillingar att skapa den digitala verkligheten. I detta arbete har denna fas delats upp de tidigare nämnda delproblemen ifrån avsnitt 1.4. Dessa delproblem visualiseras i Figur 2.2 nedanför.



Figur 2.2: Tredimensionell modell av digitala tvillingar där anskaffningen av en digital verklighet har delats upp i de delproblem som identifierats i avsnitt 1.1

2.2 Framtagning av simuleringsmiljö

För att den digitala verkligheten ska kunna efterlikna och simulera den fysiska verkligheten krävs en fotorealistisk simuleringsmiljö som möjliggör för representationer av tidsvarierande moment. Därav kan detta delproblem delas upp i följande två

huvudprocesser: fotorealistisk modellering av den fysiska verkligheten och uppbyggnad av en dynamisk simuleringsmiljö. Dessa två processers utformning kan variera utifrån valet av metod för 3D-rekonstruktion respektive det mjukvaruprogram som den digitala verkligheten sedan ska implementeras i, vilket detta avsnitt kommer att klargöra.

2.2.1 Det befintliga systemet

För att kunna fatta sunda beslut kopplat till hur problemet bäst angrips krävs grundläggande förståelse för hur AMR systemet fungerar. Enligt [1] styrs flottan av robotar från en centraliserad enhet som beräknar lokalisering, perception och banplanering. Den centrala enheten utgörs av ett datorkluster som fattar beslut utifrån sensorer placerade på AMR:er och fabrikens infrastruktur. Lokaliseringen baseras huvudsakligen på sensorer i form av kameror som är placerade i taket på fabriken. Utifrån kameraflödet lokaliseras AMR:er med hjälp av Apriltags[1] och semantisk segmentering. Utifrån vardera AMR:s position beräknas en körväg som skickas till till roboten. AMR:en exekverar körvägen med stöd en enkel kontrollenhet och feedback från en odometer.

2.2.2 Introduktion till Unreal Engine 5

Unreal Engine 5 (UE5) är en realtidsbaserad 3D-motor som tillåter rendering och simulering av interaktiva virtuella 3D miljöer [2]. Motorn bygger mestadels på det objektorienterade programmeringsspråket C++ och med den tillkommer Unreal Editor som utgör ett grafiskt användargränssnitt för utveckling av projekt. Varje projekt i UE5 innehåller en instans av klassen World, vilket fungerar som en behållare för alla instanser av klassen Levels. Dessa instanser av klassen Levels utgör i sin tur allt som ska ingå de virtuella 3D miljöer såsom statiska geometrier, interaktiva objekt och spelmekaniska komponenter.

I användargränssnittet Unreal Editor finns flertalet verktyg. Ett av dessa verktyg är Unreal Engines Blueprint Visual Scripting System, även kallad Blueprints, ett visuellt skriptsystem baserat på ett nodgränssnitt, se Bilaga för exempel A.1. Blueprints ger möjligheten att skapa objektorienterade spelmekaniska komponenter som bygger upp funktionaliteten inuti miljön. Därmed kan Blueprints underlätta utvecklingsprocessen genom att minimera användning av programmeringsspråket C++.

Vidare stödjer UE5 nedladdning av ett flertal tillägg från såväl utvecklarna som tredjepartsaktörer. Dessa tillägg kan användas för att skapa en mer verklighetstrogen simuleringsmiljö. Tillägg kan vara exempelvis föremål, texturer, animationer eller verktyg. De tillägg som nyttjas i detta projekt kommer att behandlas senare i avsnitt 3.5.1.

[1] https://docs.wpilib.org/en/stable/docs/software/vision-processing/apriltag/apriltag-intro.html

[2] `https://dev.epicgames.com/documentation/unreal-engine/foundational-knowledge-in--unreal-engine?application_version=5.6`

För att realisera informationsflödet från den digitala verkligheten till den fysiska, som illustreras i Figur 2.1, krävs det att UE5 tillåter digital bildtagning av den utvecklade simuleringsmiljön. I syfte att möjliggöra denna funktionalitet använder sig UE5 även utav ett renderingssystem bestående av följande separata trådar: Game Thread, Render Thread och RHI Thread som alla har olika ansvarsområden. I ett tidigare arbete [3] användes tillägget Movie Render Queue som producerar högkvalitativa renderingar av miljön. Tillägget använder sig utav UE5 Movie Render Pipeline [3], som bygger vidare på det ordinarie renderingssystemet, som enligt utvecklarna kan leda till bättre resultat än traditionella realtids renderingsmetoder. Dessvärre identifierade [3] denna metod som en utav flaskhalsarna i den digitala bildtagningsprocessen. Då realtids rendering av den digitala miljön anses vara en viktig funktionalitet hos den dynamiska simuleringsmiljön för nästkommande delproblem: lokalisering av objekt och styrning av objekt där arbetet avser att optimera bildtagningsprocessen enbart med det hjälp av det ordinarie renderingssystemet.

2.2.3 Sammanfattning

Övergripande sett behandlar detta delproblem hur en fotorealistisk 3D modell av verkligheten kan skapas för att sedan användas som simuleringsmiljö. Fokuset ligger i första hand på hur anskaffningen av 3D modellen, där 3D-rekonstruktions metoden 3D Gaussian Splatting kommer att användas. Sekundärt identifierades hur en simuleringsmiljö kan byggas upp sådant att den kan simulera de tidsvarierande moment som kan tänkas ske i den fysiska verkligheten. Detta kommer därav utföras med hjälp av 3D-motorn UE5. Den fotorealistiska simuleringsmiljön kommer sedan att användas för att lokalisera intressanta objekt i den skapade miljön. Avslutningsvis kommer detta delproblem även att behandla möjligheter och begränsningar med användning 3D Gaussian Splatting för att skapa en realistisk simuleringsmiljö.

2.3 Lokalisering av objekt

För att lokalisera och detektera objekt i simuleringsmiljön kan ML-modeller som använder sig av segmentering användas. Utformningen av arbetsprocessen kopplat till detta delproblem beror till stor grad av vilken segmenteringsmetod som används, vilket detta avsnitt kommer att omfatta.

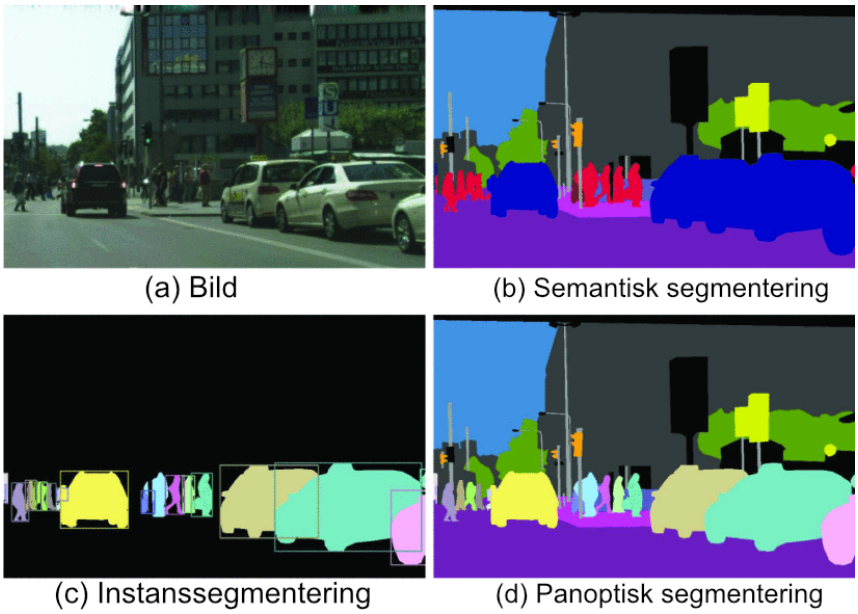
2.3.1 Introduktion till segmentering

För att kunna detektera och lokalisera objekt i simuleringsmiljön utifrån tak-kamerornas bildflöde, kan segmentering tillämpas. Segmentering innebär att varje pixel i en bild tilldelas en klass, vilket möjliggör både klassificering och lokalisering av objekt i en bild [6]. Inom området särskiljs det ofta mellan ting, som avser amorfa områden [4] såsom golv, vägg och tak, och saker som representerar individuella objekt såsom

[3] `https://dev.epicgames.com/documentation/unreal-engine/movie-render-pipeline-in-unreal-engine?application_version=5.6`

[4] Amorfa områden syftar till oregelbundna områden som inte har en fast struktur eller geometri.

människor och robotar [7]. Dessa uttryck är centrala för att förstå skillnaden mellan följande segmenteringsmetoder: instanssegmentering, semantisk segmentering och panoptisk segmentering, se Figur 2.3, som detta avsnitt avser jämföra.



Figur 2.3: I figure visas de olika segmenteringsmetoderna. Bild (a) visar orginal bild, (b) visar en semantiskt segmenterad bild, (c) instanssegmenterad bild och (d) panoptisk segmenterad bild. Bilden är anpassad från Kirillov et al. [8] © [2019] IEEE.

2.3.2 Semantisk segmentering

Vid semantisk segmentering tilldelas varje pixel i en bild en klassetikett. En klassetikett är ett ID-nummer som representerar en specifik klass, vilket möjliggör för både klassificering och lokalisering av objekt. Metoden hanterar både saker och ting, där saker saknar instansskillnad mellan de individuella objekten i bilden [7]. Detta medför i sin tur att flera objekt av samma klass, exempelvis flera personer, behandlas som en enhet. Resultatet av semantisk segmentering representeras som en bild där varje pixel är kopplad till en klassindex, vilket ofta visualiseras genom att varje klassindex tilldelas en unik färg. På så sätt kan en segmenteringskarta skapas och om nödvändigt placeras över originalbilden för att tydliggöra den semantiska indelningen [6].

Fördelen med semantisk segmentering är att amorfa områden behandlas som ett sammanhängande objekt. Områden som inte behöver räknas såsom golv, vägg eller tak, överlappas eller täcks ofta av andra objekt. Semantisk segmentering kan ändå hantera dessa separerade delar gemensamt och identifiera dem som samma typ av område. Dessvärre leder detta till att metoden saknar förmåga att särskilja individuella objekt inom samma klass. Därav anses denna metod vara mindre lämplig då funktionaliteten, att kunna skilja mellan exempelvis flera människor i samma scen, är viktig.

2.3.3 Instanssegmentering

Instanssegmentering innebär att varje pixel i en bild tilldelas en klassetikett och varje objekt tolkas som en instans [7]. Således identifieras objekt enbart som saker där dessa hanteras som instanser, vilket urskiljer sig ifrån semantisk segmentering. Detta tillåter instanssegmentering att särskilja mellan flera objekt av samma objekttyp, exempelvis flera människor som befinner sig i samma bild. Till skillnad från semantisk segmentering möjliggör instanssegmentering, utöver lokalisering, summering av varje instans i en bild [6]. Instanssegmentering kan därmed ses som en kombination av semantisk segmentering och objekt detektering, då den både klassificerar och särskiljer mellan individuella objekt.

2.3.4 Panoptisk segmentering

Panoptisk segmentering är en segmenteringsmetod som kombinerar både semantisk och instanssegmentering genom att varje pixel tilldelas en semantisk klass samtidigt som separerbara objekt identifieras som instanser och tilldelas ett instans-ID [9]. Följaktligen identifieras objekt som ting och saker, där saker likt instanssegmentering detekteras som instanser. Panoptisk segmentering är därför användbart i implementeringar som har behov av att både segmentera alla objekt av samma objekttyp som en enhet och att få ut information om varje enskild instans inom varje enhet. Samtidigt kan amorfa områden såsom golv behandlas som ett objekt, vilket är fördelaktigt om det förekommer objekt som separerar golvet i flera bitar som gör att det istället skulle kunna tolkas som flera instanser.

2.3.5 Val av segmenteringsmetod

Semantisk segmentering valdes som huvudsaklig metod eftersom arbetet främst fokuserar på pixelvis klassificering av objekt snarare än identifiering av individuella objektinstanser. Metoden är väl lämpad för att segmentera sammanhängande och amorfa områden, såsom golv, väggar och tak, där individuella instanser saknar betydelse. Därtill används semantisk segmentering brett för objektigenkänning [10][11][6]. Vidare används det även i ett tidigare arbete [3] som detta arbetet bygger vidare på, vilket gör semantisk segmentering till ett naturligt val.

Instanssegmentering undersöktes även för att analysera möjligheten att särskilja individuella objekt inom samma klass. Eftersom detta anses vara till nytta vid ett senare skede av arbetet. Eftersom styrningen, vars arbetsprocess behandlas i kapitel 5, förutsätter att särskiljning av olika objekt av samma klass är möjligt. Därför inkluderades även instanssegmentering i undersökningen.

Panoptisk segmentering övervägdes eftersom metoden kombinerar egenskaper från både semantisk segmentering och instanssegmentering. Men metoden ansågs vara för komplex i förhållande till arbetets mål. Eftersom det utvärderades att arbetet inte krävde samtidig hantering av både amorfa områden och detaljerad instansindelning, valdes panoptisk segmentering bort.

Eftersom semantisk segmentering är en väletablerad metod inom objektigenkänning och som fungerat bra i tidigare arbete [3] är det intressant att undersöka om metoden istället kan uppnå realtidssegmentering. Därutöver är det av intresse att undersöka instanssegmentering, eftersom instanslokalisering potentiellt kan bidra till styrning av AMR:er. Detta arbete jämför båda metoderna i syfte att utvärdera vilken som bättre lämpar sig för att lokalisera objekt i simuleringsmiljön.

2.3.6 Sammanfattning

Sammanfattningsvis handlar detta delproblem om hur lokaliseringen av objekt ska ske. Fokus ligger på att skapa en segmenteringsmodell för att lokalisera objekt inom fabriken. Detta kommer utföras genom datainsamling för att kunna skapa ett dataset som kan användas till träning med hjälp av överföringsinlärning. Modellerna kommer efter träning utvärderas för att hitta en passande modell för att uppnå realtids baserad inferens. De modeller som kommer utvärderas är instans segmenteringsmodellen YOLOv8-Seg Nano och semantisk segmenterings modellen Segformer-B0. Den valda segmenteringsmodellen kommer därefter att användas för att tillåta lokalisering av objekt vid styrning av AMR:er.

2.4 Styrning av autonoma mobila robotar

När perception har använts för lokalisering av simuleringsobjekt, baserat på bilddata, återstår att behandla denna information och omvandla den till styrsignaler som kan användas för att styra AMR:ers beteende. Behandlingen av data innefattar att objektens pixelkoordinater, såsom för robotar och hinder, omvandlas till världskoordinater i simuleringsmiljön. Därefter krävs ett beslutsfattande kring hur styrsignalen ska genereras, för att styra robotarna, baserat på den uppskattade omgivningen.

Det finns flera sätt att utforma denna process, beroende på hur systemarkitekturen väljs och kopplingen mellan simulering, lokalisering och styrning implementeras, vilket detta avsnitt kommer att belysa.

2.4.1 Val av systemarkitektur

För att kunna omvandla information från perception till styrsignaler krävs en systemarkitektur för att koppla samman alla ingående delar. Det finns flera sätt att utforma detta och i följande arbete har två alternativ studerats. En sluten lösning där allt sker i spelmotorn, UE5, samt en modulär lösning där ett externt ramverk kopplas till UE5.

Styrningen av AMR:erna kan implementeras som ett slutet system direkt i spelmotorn. Detta innebär att både perception och styrning hanteras inom simuleringsmiljön i UE5, med hjälp av spelmotorns inbyggda funktioner. Singh et al. [12] visar att valet av systemarkitektur påverkar realtidsprestandan av den digitala tvillingen,

där mer integrerade lösningar, som helt spelmotorbaserade simuleringar, kan reducera latens genom att minska behovet av kommunikation mellan separata delar i systemet.

Å andra sidan menar Quigley [13] att modulära systemarkitekturer möjliggör återanvändning och integration med verklig hårdvara, som används i industrin. Detta innebär att lösningar där hela systemet implementeras direkt i spelmotorn kan bli svårare att överföra till verkliga robotsystem, vilket inte är fördelaktigt vid skapandet av en digital tvilling. Samtidigt kan fenomen som vendor lock-in uppstå, vilket innebär att systemet blir beroende av den valda spelmotorns, i detta fall UE5, utveckling och stöd. Detta kan skapa problem om funktioner ändras eller om stödet för plattformen upphör.

En mer modulär lösning, som också stöds av tidigare forskning, är att använda spelmotorn enbart som en visuell representation av systemet, medan övriga funktioner hanteras externt [14]. I detta arbete skulle ett externt ramverk såsom ROS 2 eller YARP kunna ansvara för lokalisering och styrning. Däremot menar Singh et al. [12] att denna typ av lösning, med externa system, kan skapa en sämre realtidsförmåga, i form av latens i simuleringen, än den mer slutna lösningen

Däremot, till skillnad från den slutna simuleringslösningen, ger den modulära en förankring i verkligheten, eftersom system som ROS 2 används i verkliga fall inom industrin. Detta innebär att simuleringsmiljön i teorin skulle kunna bytas ut helt mot verkligheten [15]. Det blir också enklare att separera olika delsystem, som simulering, lokalisering och styrning, vilket Quigley menar att system som ROS 2 är anpassade för.

Trots att en sluten lösning kan ge lägre latens, i systemet, valdes den modulära arkitekturen, eftersom projektets syfte inte enbart var att skapa en fungerande simulering utan även att undersöka en lösning som kan kopplas till verkliga robotsystem.

2.4.2 Val av externt ramverk

För externa robotsystem finns det flera ramverk som möjliggör kommunikation mellan separata delsystem, där perception, beslutsfattande och styrning kan hanteras oberoende av varandra. I detta fall är YARP och ROS 2 vanliga exempel och har därför valts att studeras. Dessa ramverk erbjuder olika lösningar för hur data utbyts och hur systemet struktureras, vilket påverkar flexibilitet och prestanda [15], [16].

Till att börja med används både YARP och ROS 2 för att bygga distribuerade robotsystem, ett system där alla funktioner inte körs i ett program utan är uppdelade i flera mindre. Där syftet är att olika delar ska kommunicera med varandra. Där stora komplexa problem kan delas upp i färre mindre komplexa delproblem [13]. Det finns dock skillnader mellan dessa robotsystem.

YARP är även flexiblare än ROS 2 eftersom utvecklaren bestämmer fritt över kommunikationen, exempelvis stöds fler datatyper i YARP. Däremot har ROS 2 en mer standardiserad och väldefinierad struktur och har mindre utrymme för egna designval, såsom att ROS 2 har ett fast kommunikationsflöde. Detta gör bland annat YARP till ett mer öppet system, vilket kan vara fördelaktigt om full kontroll vill uppnås. [15], [16].

Ytterligare en skillnad mellan dem är att ROS 2 är mer använt inom forskning och industriella tillämpningar. Detta underlättar integrering och samarbete med dagens system. Samtidigt kan en väldefinierad struktur som ROS 2 har, till skillnad från YARP, skapa robusthet [15]. Denna standardisering kan även innebära mer dokumentation och fler tillägg till ROS 2, vilket är fördelaktigt när externa system skall kopplas samman.

Således har ROS 2 valts som ramverk för kommunikationen mellan de olika delarna, då det erbjuder en väldefinierad och standardiserad struktur samt en redan etablerad användning inom industri och forskning [15]. Genom att man kan dela upp systemet i separata delar fås ett modulärt och skalbart system där funktioner såsom perception och styrning kan utvecklas oberoende av varandra. Detta är särskilt lämpligt i fallet som projektet behandlar.

2.4.3 Val av integrationslösning mellan Unreal Engine 5 och ROS 2

För att möjliggöra simulering av kamerabaserad sensordata samt överföring av denna till ett externt ramverk såsom ROS 2 krävs ett tillägg som hanterar både datagenerering och systemstyrning i simuleringsmiljön, samtidigt som det möjliggör kommunikation mellan UE5 och det externa ROS 2-systemet. I detta arbete har två alternativ övervägts. Dessa alternativ är tilläggen RClue och Tempo. Dessa erbjuder olika tillvägagångssätt för hur data kan genereras och användas i simuleringen.

RClue är ett tillägg som används för att koppla UE5 till externa system, vilket möjliggör åtkomst till och vidare bearbetning av data från den simulerade miljön. Ramverket utvecklat av Rapyuta Robotics fokuserar endast på själva kommunikationen mellan simuleringsmiljön och externa system [17]. Generering av sensordatan behöver i detta fall utföras på annat sätt.

Tempo är en samling tillägg för UE5 som är specifikt utvecklad för simulering av robotsystem. Utöver att det möjliggör kommunikation med externa system erbjuder Tempo, till skillnad från RClue, andra verktyg för att simulera sensorer, generera bilddata och styra objekt i simuleringsmiljön, med flera. Detta betyder att Tempo är integrerat i spelmotorn och är gjort för just det ändamålet som projektet söker. Något mer som skiljer dem åt är att RClue endast har stöd för operativsystemet Linux, medan Tempo har stöd för Windows, MacOS och Linux [17], [18].

Valet blev därför Tempo, då tillägg-samlingen är en mer fullständig och integre-

rad lösning för simulering i UE5 på fler operativsystem än RClue. Detta ses som särskilt viktigt i detta arbete eftersom många funktionaliteter som simulering av sensorer och en koppling mellan ROS 2 och UE5 är nödvändiga för att skapa en realtidsbaserad digital tvilling.

2.4.4 Sammanfattning

Sammanfattningsvis behandlar detta delproblem hur information från kamerabaserad perception kan omvandlas till styrsignaler för AMR:erna. Fokus ligger på att beskriva hur bildbaserad information, i form av detekterade objekt, kan bearbetas och användas för att styra de mobila robotarna. Detta kommer att utföras med det externa ramverket ROS 2, vilket utgör systemarkitekturen. ROS 2 kommer, med hjälp av plugin-samlingen Tempo, att integreras med UE5 för att generera sensordata till lokaliseringen av objekt. Samtidigt kommer Tempo att användas för att skicka styrsignaler till de mobila robotarna. Slutligen ska även delproblemet undersöka frågeställningen kring svagheter och styrkor med att använda perception för att styra mobila robotar.

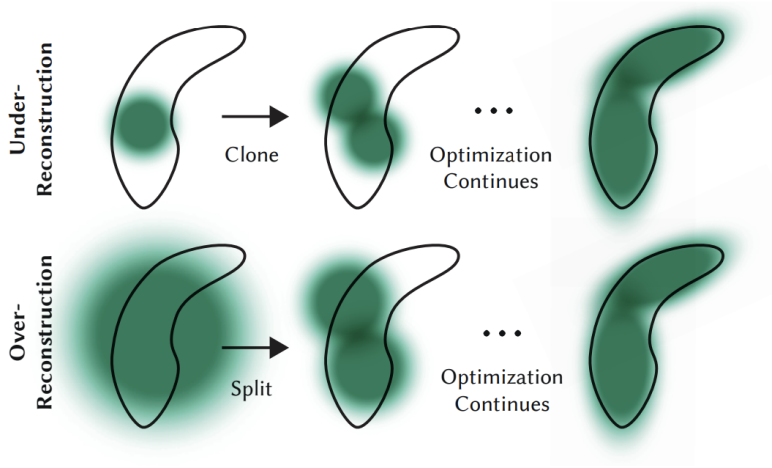
3

Framtagning av simuleringsmiljö

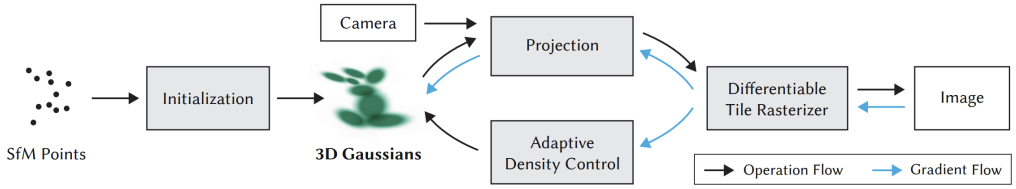
I detta kapitel beskrivs hur verkliga miljön har hämtats in och modellen av fabriken har konstruerats. Kapitlet redogör för hur simuleringen är uppbyggd visuellt samt vilka detaljer som är nödvändiga för att simuleringen ska fungera korrekt. Vidare beskrivs hur olika delar samverkar samt hur karaktärer och andra dynamiska objekt implementeras i miljön. Det visas även hur dessa objekt rör sig inom simuleringen.

3.1 Introduktion till 3D Gaussian Splatting

En 3DGS miljö byggs upp av ett moln av gaussiska ellipsoider där deras överlapp är det som skapar bilden. Metoden utgår från en uppsättning av sammanhängande bilder av en miljö. Utifrån dessa skapas ett glest punktmoln genom Structure from motion (SFM). Det glesa punktmolnet fungerar som en initial placering av de gaussiska ellipsoiderna som var och en uttrycks av en position, en opacitet, en rotation, en skala och en Klotytefunktion. Klotytefunktionen beskriver den individuella gaussens färg utifrån vilken vinkel som gaussen betraktas ifrån. Miljön av initialt placerade gausser är gles och behöver förbättras genom en iterativa process som förfinar den etablerade miljön. Processen går ut på att dela, omforma och omplacera nya samt befintliga gausser så att miljön närmare efterliknar grundmaterialet [19]. I denna process gallras även redundanta gausser vid var 100:e iteration om deras opacitet understiger ett givet gränsvärde. För att algoritmen ska kunna ta ett beslut om vilken typ av optimering som skall ske renderas först miljön i befintligt skick. Bilden jämförs med underlaget och skillnaden mellan dem resulterar i en gradient för varje gauss. Denna gradient skickas bakåt i processen till en adaptiv densitets kontroll som avgör om ett område av gausser är för underanpassat eller överanpassat. Är ett område underanpassat klonas gaussen och dennes form justeras. Är området överanpassat skalas gaussen först ner för att sedan splittras. Denna delprocess beskrivs visuellt i Figur 3.1. Optimeringsprocessen redogörs sammanfattningsvis av Figur 3.2



Figur 3.1: Representation av adaptiv densitet kontroll klonar alternativt splittrar gausser beroende på passform, [19].



Figur 3.2: Optimiseringsprocessen av en 3DGS miljö, [19].

3.2 Datainsamling

Följande beskriver de metoder och den information som ligger till grund för insamlingen av data till simuleringsmiljön. Vid utveckling av en simuleringsmiljö är en välstrukturerad och tillförlitlig datainsamling avgörande. Därför redogörs hur data har samlats in samt vilka verktyg och metoder som har nyttjats.

3.2.1 Kamera för datainsamling

För insamling av data till 3DGS-miljö har en kombinerad vidvinkel och lidar kamera nyttjats av typen Xgrids PortalCam[1]. Kameran är handhållen och möjliggör skapandet av fotorealistiska 3D-modeller baserat på 3DGS. Kameran kan användas för att fånga in hela miljöer så väl som enskilda objekt. Kameran är utrustad med en Light Detection and Ranging (Lidar) sensor samt fyra vidvinkel kameror. En Lidar-sensor mäter avstånd med hjälp av en laser och kan utifrån det skapa ett punktmoln[20]. Huvudsyftet med kamerans Lidar-sensor är att skapa ett tredimensionellt punktmoln för att utgör stöd för navigering i data samt orientering av de Gaussiska Splattarna. För Lidar sensorns specifikationer se Appendix A.1 Genom

[1]https://www.xgrids.com/intl/portalcam

en kombination av dessa komponenter kan kameran registrera både färg- och djupinformation simultant. Denna information används därefter för att återskapa en navigerbar 3D-modell av omgivningen [2].

3.2.2 Arbetssätt för datainsamling

I syfte att erhålla en högkvalitativ 3DGS-modell följdes tillverkarens riktlinjer för inspelning[3]. Riktlinjerna presenteras i tabell 3.1. AB Volvos fabriker i Tuve har högt i tak, vilket medför utmaningar vid datainsamlingen då det försvårar att filma miljön från flera perspektiv. För att möjliggöra datainsamling nära taknivå användes ett utfällbart kamerastativ. Syftet med denna var att bättre fånga perspektivet från de befintliga kamerorna.

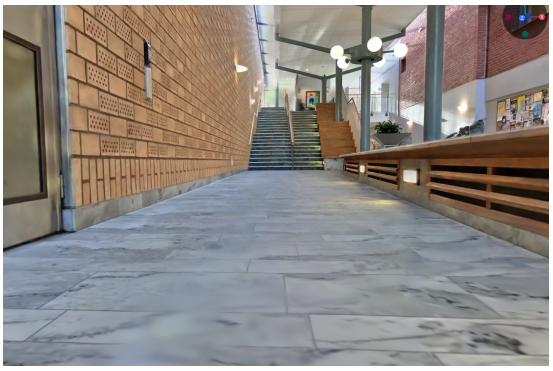
Tabell 3.1: Rekommenderat tillvägagångssätt vid inspelning med XGrid PortalCam.

Att föredra	Att undvika
• Filma med en gånghastighet på under 0,5 m/s. • Fånga miljön från flera vinklar och riktningar. • Röra sig slalom för högre punktmålsdensitet. • Avsluta inspelningen med kameran stående på marken.	• Filma föremål på avstånd under 0,5 m. • Filma föremål med hög glans på avstånd under 1 m. • Luta kameran mer än 60° upp eller ner.

Utöver dessa riktlinjer prövades även nya metoder för att tackla problem som tidigare arbeten påträffat. I det föregående arbetet beskrivs ett fenomen där 3D gausser vid golvet fångats upp med lägre precision vilket lett till att föremål som placeras i miljön framstår som suddiga eller nedsjunkna i golvet[3]. För att förebygga detta prövades att filma nära golvet mot horisonten.
Figur 3.3 och 3.4 visar skillnaden mellan att filma nära golvet och inte. Resultatet påvisar att metoden gynnar golvets utseende. Av denna anledning tillämpades metoden vid filmningen av fabriken.

[2] `https://www.laserscanning-europe.com/en/xgrids-portalcam`
[3] https://www.xgrids.com/intl/support/tutorials?page=PortalCam

Figur 3.3: Inspelning tagen i brösthöjd.



Figur 3.4: Inspelning tagen i golvhöjd.

3.2.3 Bildtagning av Volvo Lastvagnars fabrik i Tuve

Under två olika tillfällen spelades ett område på fabriken i Tuve in. Detta i syftet att utvärdera hur olika inspelningstekniker påverkar kvaliteten på 3DGS-miljön. Inspelningen omfattade ett område på ca $300m^2$ vars totala omfattning redogörs i figur 3.5.

Inspelning A

Under inspelning A filmades hela området som visas i figur 3.5 i en tagning. Inspelningen tog ca 60 minuter. Vid inspelningen följdes rekommendationerna givna i tabell 3.1 med två undantag. Dels avslutades inte inspelningen med kameran stående på marken, dels startade och slutade inte inspelningen på samma ställe. Vid tillfället spelades även föremål av särskilt intresse in. Bland annat en AMR robot och en gaffeltruck.

Inspelning B

Vid inspelning B spelades ett mindre område in vilket resulterade i en inspelnings tid på ca 30 min, se figur 3.5. Denna gång följdes rekommendationerna till fullo och särskild vikt lades vid att börja och sluta på samma plats.

Figur 3.5: Översiktsbild av området i fabriken som spelades in vid inspelning A. Inspelning B begränsades till området markerat i rött.

3.3 Generering av 3D Gaussian modell

I detta avsnitt beskrivs arbetsflödet för generering av en 3DGS-modell baserat på datan som samlats in enligt kap. 3.2. Avsnittet behandlar även rendering samt vilken programvara och vilka verktyg som nyttjas.

3.3.1 Användning av renderingsprogram (LCC)

För att skapa 3DGS-miljöer utifrån den verkliga miljön bearbetades inspelningarna från kameran i den tillhörande programvaran Lixel CyberColor[4](LCC). Efter behandling kan miljön exporteras i 3DGS formaten `.lcc` samt `.ply`. Filformat `.lcc` är ett särskilt format utvecklat av XGRIDS för att reducera filstorleken jämfört med andra format och bevara visuell kvalité[5]. Filformatet är inte lika vedertaget som `.ply` eftersom formatet har begränsad kompatibilitet utanför XGRIDS ekosystem. Däremot erbjuder formatet särskilda fördelar, till exempel den mindre filstorleken, vilket gör det mer lämpat för somliga applikationer.

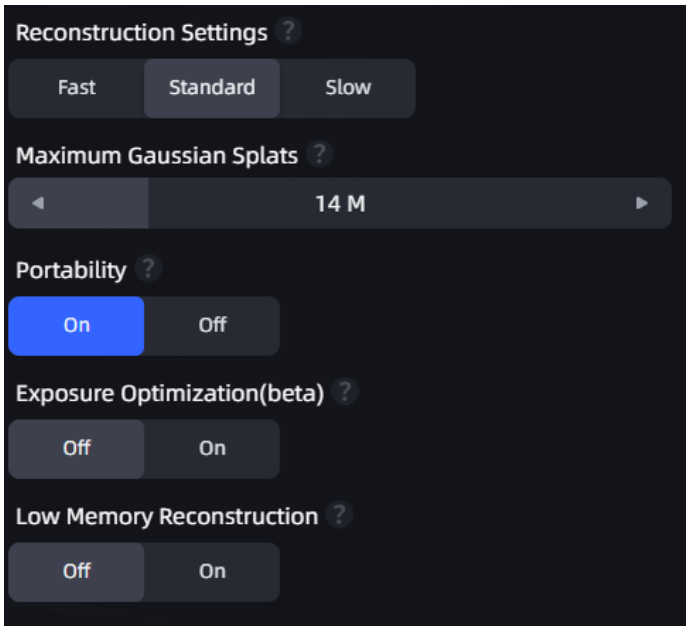
Under projektets gång nyttjades formatet `.lcc` primärt. Motiveringen till detta var att förebygga ett antal problem som påträffades i ett tidigare arbete [3]. Där fann man att det uppstod förluster vid import/export mellan program. För att bevara kvalitén av modellen användes tillverkarens egna tillägg till UE5. Initialt var även .lcc det enda tillgängliga exportalternativet från LCC. Detta innebar att det inte var möjligt att nyttja något annat tillägg. Formatet `.ply` tillgängliggjordes först senare under projektets gång, varpå det bedömdes vara alltför omfattande att övergå till formatet. En bidragande faktor till denna bedömning var även bristen på tillförlitliga och fungerande tillägg för `.ply` i UE5. Beprövade tillägg för detta format finns tillgängliga för andra versioner av spelmotorn, men ytterst få var kompatibla med version 5.6 som användes.

I LCC-studios kan olika inställningar justeras beroende på önskad kvalitet och tillgänglig beräkningskraft. Processen är framförallt mycket krävande på datorns RAM minne samt Grafikkortet och dess Video RAM. Inspelningarna som beskrivs i kapitel 3.2 behandlades till en början med standard alternativ vilket gav följande nyckeltal: Renderingstiden för 60 minuters inspelningen översteg 12 timmar, de kortare inspelningarna resulterade i snabbare men fortfarande betydande beräkningstider på 4-6 timmar. För att undersöka hur den färdiga miljön påverkas av inställningarna renderades en av de kortare inspelningarna med tre olika parametrar. Parametrarna som kan justeras framgår i Figur 3.6. För samtliga renderingar var Reconstruction Settings tillika Slow, Maximum Gaussians Splats justerades, övriga parametrar var inställda enligt Figur 3.6. Vid rendering prövades 14 miljoner, 30 miljoner och 50 miljoner Maximum Gaussian Splats. Skillnaden i miljöns utseende kan iakttas i Figur 3.7. Utifrån figuren kan viss skillnad mellan renderingarna urskiljas. Tydligast skillnad observeras på skylten ”19-51-2”, mitt i bild, där skärpan blir något bättre med större antal gausser. Denna skillnad kan vara av nytta i somliga användnings-

[4]https://xgrids.com/intl/lcc

[5]https://www.xgrids.com/intl/support/download?page=LCCStudio]

områden till exempel om det är av vikt att läsa av skyltar i miljön. För detta arbete anses 30M gausser vara tillfredsställande för simuleringens ändamål.



Figur 3.6: Parametrar för rendering av 3DGS-miljö, de aktuella parametrarna är standard

Efter rendering exporterades även tillhörande mesh, benämns polygonytor i denna rapport, i form av `.obj` filer. Dessa behandlades i programmet Blender. Blender är ett open-source program för digital formgivning som erbjuder en mängd verktyg för att redigera 3D modeller[6]. Det finns primärt två anledningar till varför polygonytorna behöver behandlas. Den främsta anledningen är att kontrollera att polygonytan är av tillräckligt god kvalité. Den ursprungliga polygonytan kan innehålla oönskade detaljer som hål eller missformade föremål, i Blender kan dessa retuscheras bort. Den sekundära anledningen är att överflödiga detaljer och områden som inte kommer vara del utav simuleringen kan tas bort i syfte att optimera simuleringen.

Utifrån Figur 3.8 tycks antalet gausser ha liten till ingen påverkan på kvalitén av polygonytan. Det som däremot har större påverkan på polygonytans kvalité är de inställningarna som nyttjas vid export och import mellan Blender och UE5. Vid export är de mest vitala att Apply modifiers är på, annars finns risk att modifikationer av polygonytan inte medföljer till UE5. Vid import till UE5 nyttjades inställningar i enlighet med Tabell 3.2:

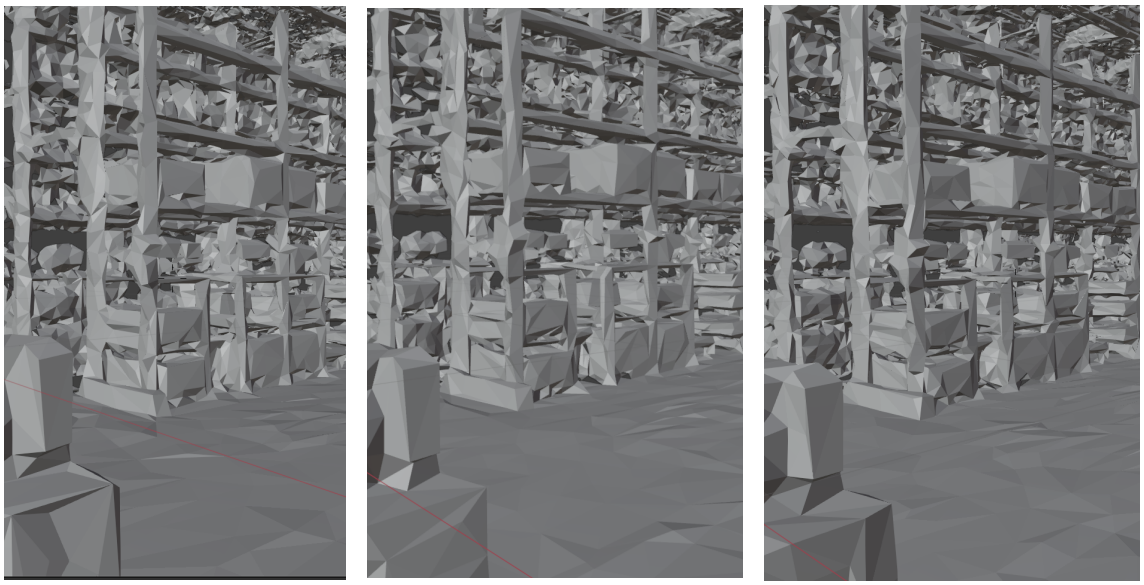
[6]https://www.blender.org/features/

Figur 3.7: Rendering av miljön med, från vänster till höger, 14M gausser, 30M gausser samt 50M gausser

Tabell 3.2: Lista över importinställningar

Inställning	**Status**
Import Meshes	På
Import Collision	Av
Recompute Normals	Av
Recompute Tangents	På
Generate Lightmap UVs	Av
Build Nanite	Av

Denna uppsättning är framtagen för att maximera polygonytans kvalitet samtidigt som lasten på datorn minimeras. Då polygonytorna för fabriken innehåller ca 700 000 hörn kan inställningar som Build Nanite orsaka att importen blir för tung och att spelmotorn kraschar.



Figur 3.8: Resulterande polygonytor utifrån olika nivåer av 3DGS kvalité. Från vänster till höger: 14M gausser, 30M gausser samt 50M gausser

3.4 Uppbyggnad av simuleringsmiljö

I detta avsnitt redogörs för hur miljön i spelmotorn konstrueras och de program och verktyg som använts beskrivs. För att åstadkomma en realistisk simuleringsmiljö behöver flertalet aspekter beaktas. Särskilt viktiga aspekter inkluderar formgivning, färgsättning och ljussättning. Detta avsnitt beskriver även hur geometriska komponenter möjliggöra fysikalisk interaktion samt hur förutsättningar för automatisk generering av data beaktas.

3.4.1 Visualisering av 3DGS modell i UE5

För att implementera 3DGS-modeller i simuleringsmiljön nyttjas de filer som producerats i LCC tillsammans med ett tillägg för rendering av Gaussian Splats. Tillägget som tillämpades var LLC4Unreal [7] och är utvecklat av Xgrid för att importera 3DGS-filer i formatet .lcc. Tillägg möjliggör integrering av 3DGS modeller i UE5 samt tillhandahåller ytterligare verktyg och inställningar för hantera dessa.

För att importera en 3DGS i modell nyttjades komponenten LCCActors från tillägget LLC4Unreal. Komponenter av typen LCCActor huserar en filsökväg till den 3DGS som ska användas samt inställningar och verktyg som påverkar miljöns utseende. Ett av dessa verktyg benämns LCCClippingVolume. En LCCClippingVolume utgör en volym i spelmotorn som nyttjas för att tar bort gausser som antingen befinner sig innanför eller utanför volymen. I simuleringsmiljön användes verktyget för att klippa bort de delar av fabriken som inte syntes under simuleringen i syfte att spara prestanda.

[7] `https://developer.xgrids.com/#/download?page=LCC_UNREAL_SDK_UE56`

I Figur 3.9a, 3.9b och 3.9c observeras viss försämring av kvalitet mellan programmen. Däremot bibehölls fortfarande en hög visuell nivå. Vid jämförelse med den verkliga bilden framgår det att miljöerna är mycket lika, vilket var målet med implementationen.

(a) Bild på fabriken tagen i verkligheten

(b) Bild på fabriken tagen i LCC.

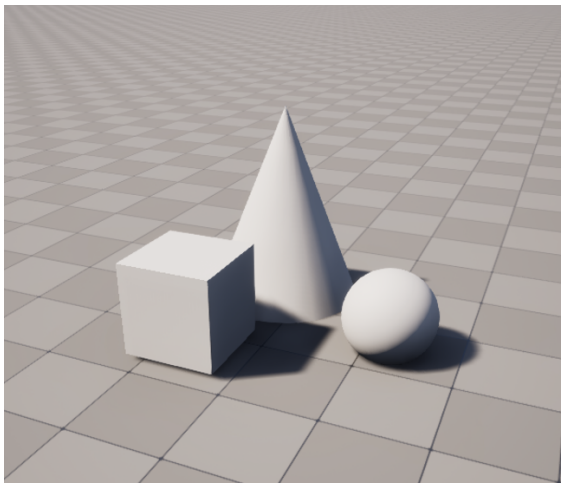
(c) Bild på fabriken tagen i UE5.

Figur 3.9: Jämförelse av fotorealism mellan bild tagen i verkligheten, LCC och UE5

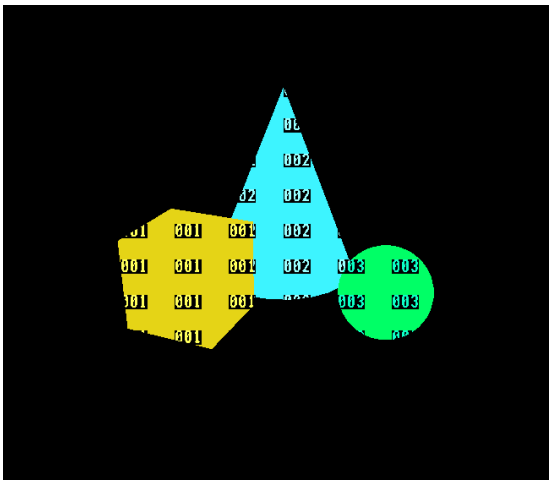
3.4.2 Integrering och skapandet av polygonytor

I UE5 används polygonytor, även känt som mesh, för att beskriva ett objekts geometri och tillåta diverse egenskaper[8]. Exempel på dessa är texturer och kollision med andra objekt. En polygonyta är en tredimensionell modell av ett objekt. För att skapa objektet används ytor, kanter och hörn. Hörn är unika punkter i det tredimensionella rummet som förbinds av kanter. Ytorna kallas polygoner och bildas när kanterna omsluter ett område. Dessa är oftast trianglar eller fyrhörningar.

3DGS-modeller var inte tillräckligt för att skapa en fungerade simulerings i UE5. För detta krävdes även polygonytor. Bland annat möjliggjorde de att karaktärer kan röra sig i miljön. För senare del av projektet, se kap 4.1.1, är det av särskild vikt att kunna särskilja golv och väggar. På grund av detta ersätts det befintliga golvet som skapas av LCC med en polygonmodell i from av ett plan. Detta möjliggör att golv och väggar ses som två olika objekt. Polygonytor i UE5 kan associeras med olika värden, så kallade CustomStencil ID, i spannet 0-255. En särskild rendering kan då göras där RGB värdet för varje pixel ersätts med värdet för CustomStencil ID. Denna särskilda rendering resulterar i en maskning av den ordinarie bilden, se Figur 3.10b.



(a) Polygonmodeller sedd i RGB rendering



(b) Mask av polygonmodeller sedd i CustomStencil rendering

Figur 3.10: Förevisning av hur polygonytor kan associeras med CustomStencil ID

3.4.3 Ljus och färgsättning

Då 3DGS miljöer importeras till UE5 medföljer en typ av ljussättning som kallas Tone map. Denna ljussättningen är särskilt anpassad för just den specifika splatten och gör att den efterliknar den verkliga miljön mycket nära. Problem uppstår däremot när komponenter som inte utgörs av splattar ska visualiseras parallellt. Då tone mapping appliceras på alla komponenter i miljön resulterar det i att ljussättningen

[8]https://www.techtarget.com/whatis/definition/3D-mesh

för allt utöver splattar blir mycket förvrängd. Av denna anledning kan tone mapping inte användas utan ljussättningen behöver justeras manuellt för ett efterlikna verkligheten. För detta används tre komponenter av typen DirectionalLight som ljussätter nivån. Vanligtvis nyttjas inte flera instanser av DirectionalLight, istället nyttjas andra typer av ljuskomponenter för att skapa en realistisk ljusbild. På grund av hur bilderna i ett senare skede fångas in, se avsnitt 4.1.1, fångas bara direktriktat ljus upp. Detta gör det tvunget att använda flera direkta ljuskällor för att uppnå ett realistiskt utseende.

Tone map justerar även splattens färgsättning och utan den behöver det justeras manuellt. För att åstadkomma detta anpassas splattens mättnad, kontrast och gamma tills dess att splatten är verklighetstrogen. Då andra karaktärer adderas till simuleringen behöver även deras färgsättning justeras för att minska sim-to-real gapet. Utan justering blir kontrasten mellan 3DGS komponenter och polygonytor baserade komponenter särskilt tydlig. För att kunna justera färgsättningen av icke 3DGS komponenter adderas en PostProcessingVolume till simuleringen. Denna volym påverkar allt utom 3DGS komponenterna och erbjuder en mängd inställningar för utseende. Bland annat inställningar för färgsättning som justeras på samma vis som för splatten.

3.5 Simulering av dynamiska händelser

Vid skapandet av en verklighetstrogen fabrikssimuleringsmiljö behövs dynamiska aktörer, för detta arbete är människor och AMR:er av särskilt intresse. Detta avsnitt beskriver hur dessa komponenter importeras och integreras i miljön, samt hur olika scenarier kan tillämpas.

3.5.1 Simulering av människor

För att uppnå en realistisk representation av fabriksmiljön och dess dagliga verksamhet krävs att människor inkluderas och kan förflytta sig naturligt i simuleringen. Ett steg i den riktningen är att skapa verklighetstrogna digital människor. För detta nyttjades UE5:s inbyggda tillägg MetaHumans[9]. Med hjälp av verktyget MetaHuman Creator kan mycket realistiska karaktärer med hög detaljnivå skapas vars utseende kan justeras efter behov och ändamål, se Figur 3.11. För att anpassa karaktärerna till fabriksmiljön importerades kläder från FAB. FAB är en digital marknadsplats för nedladdning av olika digitala resurser för användning i exempelvis UE5 [10]. Dessa kläder redigerades sedan för att efterlikna arbetskläderna som vid tillfället användes på Volvos AB, se Figur 3.12.

För att MetaHumans ska kunna röra sig likt en människa i simuleringen användes en karaktär tillhörande UE5:s standardbibliotek vid namnet Third person character. Med denna inbyggda karaktär tillkommer även blueprints för rörelse och animering.

[9]https://www.metahuman.com/
[10]https://www.fab.com/

Tack vare detta kan utseendet för Third person character substitueras det av en Metahuman för att skapa en karaktär som både ser realistisk ut samt rör sig verklighetstroget. Rent tekniskt byts Third person charachter:s polygon-modell ut mot den som tillhör ens MetaHuman.

Figur 3.11: Exempel på karaktär skapad med UE5 MetaHuman Editor.

Figur 3.12: MetaHuman i arbetskläder som efterliknar verklig motpart.

3.5.2 Autonoma mobila robotar

Utöver människor utgör AMR:er en vital del av fabriken. För att återskapa AMR:er i simuleringsmiljön användes en 3DGS-modell insamlad vid inspelning A, se avsnitt 3.2.3. För att senare kunna kontrollera AMR:ens rörelse skapades en ny karaktär klass för robotarna. Karaktärklassen innehåller en polygon-modell av AMR:en som togs fram utifrån den verklig AMR:ens CAD-modell. Polygon-modellen är även av vikt för att senare kunna skapa träningsdata med hjälp av CustomStencil ID. AMR klassen och 3DGS-moddellen lades in som separata komponenter i nivån där 3DGS-modellen placerades under karaktären i nivåns outliner hierarki. På så vis kopplades positionen av 3DGS modellen ihop med karaktärens. Därefter justeras 3DGS-Modellens placering så att den överlappade fullständigt med polygon-modellen varpå polygon-modellen doldes. Totalt skapades tre olika typer av AMR:er. Den första typen navigerar slumpartat i miljön. Den andra navigerar sekventiellt mellan hållpunkter i miljön. Den tredje typen kommunicerar med ROS 2 och kan styras där igenom.

3.5.3 Autonom navigering

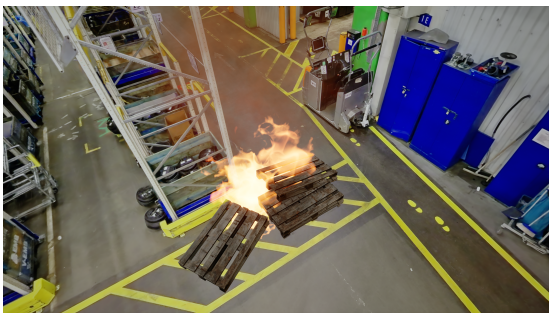
För att underlätta inhämtning av träningsdata är det av vikt att karaktärerna i simuleringen kan röra sig autonomt. I det syftet skapades tre möjliga sätt för karaktärerna att röra sig på. Två metoder involverar blueprints och kommer redogöras för här. Den tredje metoden bygger på styrning via tillägget Tempo och förklaras närmare i avsnitt 5.4.3. De karaktärer som rör sig med hjälp av blueprints rör sig utifrån en automatisk genererad navigationsyta, i UE5 kallat navmesh. Denna konstrueras utifrån de statiska polygonytorna och en komponent som heter NavMeshBoundsVolume. Volymen appliceras i nivån på de områden där karaktärerna ska röra sig. Spelmotorn avgör sedan utifrån bland annat polygonernas lutning och storlek vilka delar av ytan som är framkomlig. Den ena blueprint metoden syftar till att simulera ett slumpartat rörelsemönster. Logiken för rörelsen bygger på funktionen AI Move-

To. Funktionen hämtar först en navigerbar koordinat i navmesh inom en radie på 100m och flyttar karaktären dit. När karaktären nått punkten eller inser att den inte är nåbar hämtas en ny punkt varpå algoritmen börjar om, se Appendix A.1. Det andra rörelsemönstret syftar till att simulera ett mer systematiskt rörelsemönster. I nivån placeras så kallade TargetPoints, de är tillgängliga i UE5s standardbibliotek och fungerar som hållpunkter. I detaljfönstret för respektive karaktär med detta rörelsemönster adderas de hållpunker som karaktären ska röra sig efter i den ordningen som eftersträvas. När rörelsen exekveras hämtar funktionen AI MoveTo kordinaterna för en hållpunkt från listan Patrol Pointsoch förflyttar karaktären dit. Därefter uppdateras index värdet för listan av koordinater. Efter det väntar karaktären i två sekunder innan den går vidare till nästa punkt i listan, se Appendix A.2

3.5.4 Ovanliga händelser

En viktig egenskap hos simuleringsmiljön är att den möjliggör simulering av udda, ovanliga och farliga fenomen som kan uppstå i fabriken. För att åskådliggöra denna egenskap har ett antal scenarion skapats i miljön. I Figur 3.13 ses hur lastpallar har importeras i miljön. Figur 3.14 visar hur lastpallar simuleras fatta eld med hjälp av UE5 inbyggda effekter. Det går även att simulera hur fabrikens kameror påverkas av till exempel smuts, damm och repor, se Appendix A.8

Figur 3.13: Lastpallar importerade till miljön.



Figur 3.14: Simulering av brinnande lastpallar.

3.6 Utvärdering

Följande diskuteras den färdiga simuleringsmiljön samt dess funktioner och brister. Det resoneras även kring huruvida delmoment bidragit till huvudsakliga mål och vilka begränsningar som kvarstår för att minska sim-to-real gapet.

I avsnitt 3.2 och 3.3 beskrivs processen av att samla in data och hur den behandlades för att skapa 3DGS-modellen. Överlag har modeller oberoende renderings kvalitet uppnått ett tillfredsställande utseende för arbetes ändamål. Däremot har vissa inspelningsmetoder resulterat i att delar av modellen blivit obrukbara.

Resultatet från inspelning A och B visas i Figur 3.15 respektive 3.16. I modellens skapad utav inspelning A uppstod dubbla geometrier, se Figur 3.17. Detta fenomen observerades inte i inspelning B, se figur3.18.

Figur 3.15: 3DGS-modell av inskanning A.

Figur 3.16: 3DGS-modell av inskanning B.

Figur 3.17: 3DGS-modell av inskanning A, som visar dubbla geometrier av miljön.

Figur 3.18: 3DGS-modell av inskanning B, som korrekt rekonstruktion av miljön.

Anledningen till att miljön fick dubbla geometrier beror med största sannolikhet på att inspelning A inte avslutades med kameran stående på marken. Detta anses vara den mest sannolika orsaken på grund av att det är det enda avvikelsen från tillverkarens rekommenderade handhavande. Andra potentiella orsaker inkluderar inspelningens längd samt att inspelningen ej startades och avslutades på samma ställe. Ingen av dessa avvikelse förekom vid inspelning B vars resulterande modell är fri från defekter. Det är även värt att poängtera att skillnaderna mellan resultaten för inspelning A och B, bortsett från de dubbla geometrierna, är mycket små.

Ett ofrånkomligt problem med sättet som simuleringsmiljön angrips är behovet av både 3DGS-modeller och polygonytor. Behovet av båda parter utgör en begränsning sett till att processen att skapa simuleringsmiljön blir mer komplicerad. Processen blir mer komplicerad på grund av att varje 3DGS modell behöver en tillhörande högupplöst polygonyta. I de fall där polygonytan som Skapas i LCC inte duger kan

det vara mycket utmanande att skapa en duglig motpart. Det är även beräkningsmässig ineffektivt att behöva simulera både splattar och Polygonytor för samma komponent. I ideala fall hade 3DGS modellen även erbjudit den funktionalitet som polygonytor medför. Potentiella alternativ som uppfyller just detta diskuteras närmare i avsnitt 7.1

Vid implementering av 3DGS-miljön i UE5 användes tillägget LLC4Unreal. Syftet med att använda detta tillägg tillhandahållet av kamerans utvecklare var att minimera kvalitetsförlusterna. Då föregående års arbete hade ett betydande problem av försämringen av kvaliteten när de importerades i UE5 [3]. Den importerade miljön bibehåller hög kvalitet, men visar på en viss kvalitetsförlust. Dock anses försämringen vara liten och därmed ej vara av stor betydelse för arbetes syfte.

Ett område där sim-to-real gapet fortfarande kan minska kretsar kring hur simuleringen ljussätts. I avsnitt 3.4.3 beskrivs hur simuleringen är begränsad till enbart direkta, statiska ljuskällor. Detta medför att simulering av dynamiska ljusförhållanden under simuleringens gång blir mycket utmanande att implementera. På grund av hur 3DGS miljöer renderas i spelmotorn finns det även kraftiga begränsningar kopplat till hur skuggor visas i miljön. Komponenter som formges av polygon-modeller kastar enbart skuggor över sig själva och andra polygonmodeller. Inga skuggor faller på 3DGS miljön vilket är en starkt bidragande orsak till sim-to-real gapet. Lösningar för att simulera dynamiska ljusförhållanden och skapa dynamiska skuggor har enbart utforskats ytligt i detta arbete utan större framgång. Lösningar till dessa problem utgör ett potentiellt område för fortsatt forskning.

En begränsning som observerades med tillägget LLC4Unreal var svårigheter att rendera flera 3DGS-modeller i samma miljö. Det visade sig vara utmanande för tillägget att rendera de olika modellerna i rätt ordning. Denna ordning är binär och innebär att en AMR som placeras mellan två väggar i splatten antingen renderas helt framför eller helt bakom väggarna. Det medförde att vid vissa vinklar syntes AMR:er genom väggarna, medan de i andra vinklar försvann helt, se Figur 3.19. Eftersom detta arbete huvudsakligen fokuserar på perspektiv ovanifrån kunde problemet kringgås genom att låta AMR:er alltid renderas först.

Figur 3.19: Felaktig renderingsordning när flera 3DGS modeller existerar i samma miljö

Avslutningsvis erbjuder den kompletta simuleringsmiljön goda förutsättningar för att skapa träningsdata och utvärdera styrning. De goda förutsättningarna är dock begränsade till robotsystem likt det som för närvarande finns i fabriken. Anledningen till detta är att flera av problemen som påträffades kunde kringgås tack vare det befintliga systemets utformning. Nästa steg i att skapa en digitala verkligheten är förmågan att identifiera simuleringens delar med hjälp av segmentering

4
Lokalisering av objekt

Simuleringsmiljön som byggdes upp i kapitel 3 kan sedan användas för att lokalisera objekt, såsom de tidigare nämnda karaktärerna i avsnitt 3.5, med hjälp av maskininlärningsmodeller (ML-modeller) som bygger på segmentering. För att både träna och sedan använda ML-modellerna till detta syfte krävs först stöd för digital bildtagning av simuleringsmiljön. Detta för att dels möjliggöra för insamling av data samt inferens i ML-modeller. På grund av det här redogörs för en metod som möjliggör digital bildtagning av simuleringsmiljön i UE5. Därefter presenteras arbetsprocessen för hur den insamlade datan ifrån bildtagningsprocessen tillsammans med instans- och semantisk segmentering används av förutbestämda ML-modeller under övervakad inlärning, för att möjliggöra lokalisering av objekten. Till sist kommer resultaten från dessa ML-modeller att jämföras mot varandra.

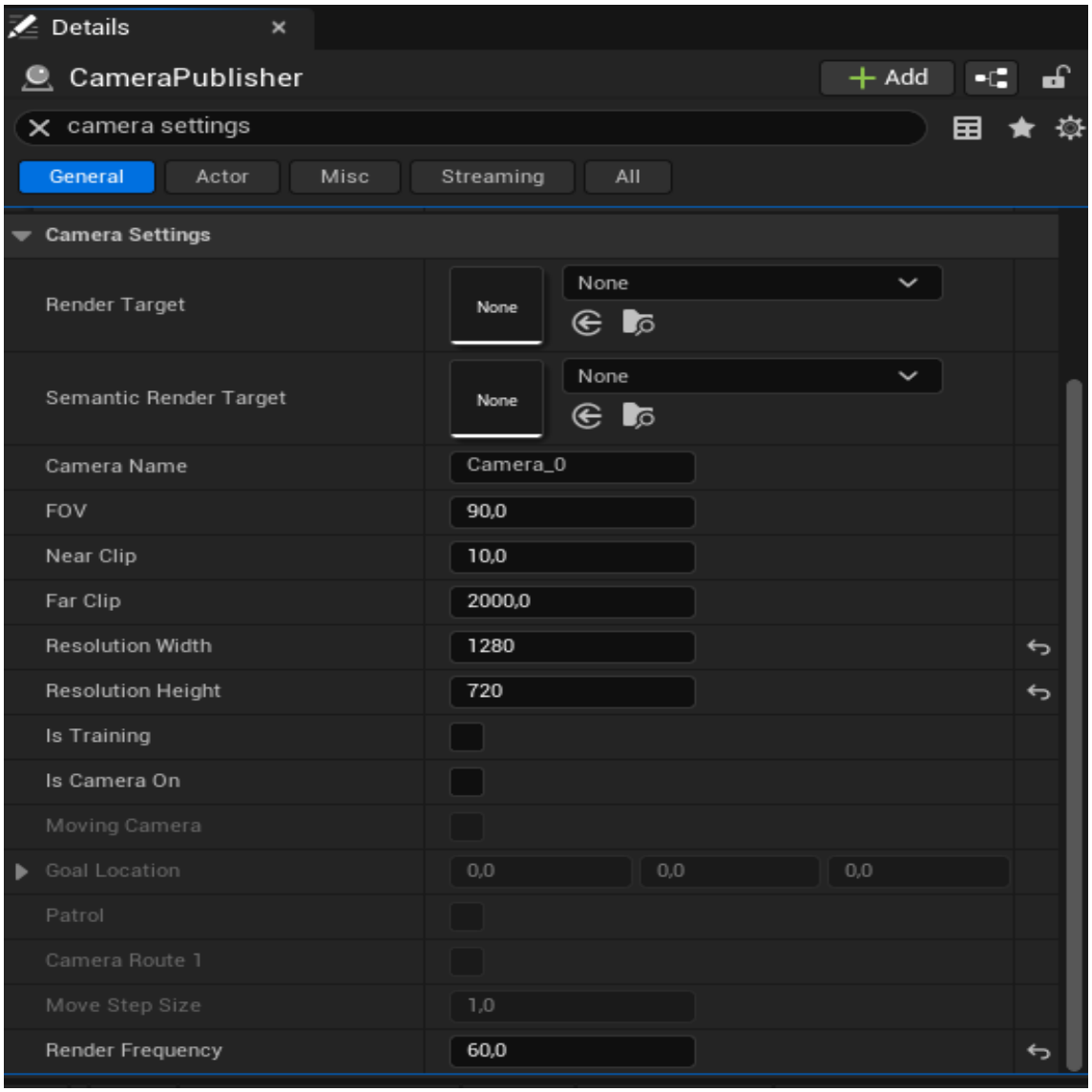
4.1 Digital avbildning av simuleringsmiljö

Under övervakad inlärning av ML-modeller krävs stora mängder annoterad träningsdata. I och med det här introduceras ett flertal olika utmaningar, varav en av de är genereringen av data. Traditionellt sett görs själva annoteringen av data manuellt vilket kan vara en resurskrävande process [11]. Därav kommer detta avsnitt att presentera en alternativ metod för digitalt bildförvärv av simuleringsmiljön och användning av bilddata för inferens av ML-modeller. Efter att en mängd data genererats kan det vara nödvändigt att behandla den för att öka robustheten hos ML-modellerna. Följaktligen kommer även metoden dataökning och efterbearbetning av data att beskrivas.

4.1.1 Syntetisk datagenerering

Som alternativ till den tidigare nämnda resurskrävande processen kan generering av syntetisk annoterad träningsdata i 3D-motorn UE5 underlätta. Med avsikt att kunna rendera samtliga delar av den spelbara ytan, se avsnitt 3.5.3, har därav digitala kamerainstanser placerats ut i simuleringsmiljön. Dessa kameror är av objekttyp Actor och bygger på UE5 klasserna USceneCaptureComponent2D och UTextureRenderTargetFormat som tillåter för rendering av scener i simuleringsmiljön samt konstruktion av behållare med valfri textur, även kallad Render Target. Arbetsprocessen kring den digitala bildtagningen av fabriksmiljön kan justeras utefter ändamål. Därav har de digitala kamerorna två olika driftlägen varav följande är: träningsläge och simuleringsläge. Huruvida vilket av driftlägena som var aktivt un-

der datainsamlingen bestämdes av Is Training variabeln som kan manipuleras i kamerans användargränssnitt, vilket ses nedan i Figur 4.1.

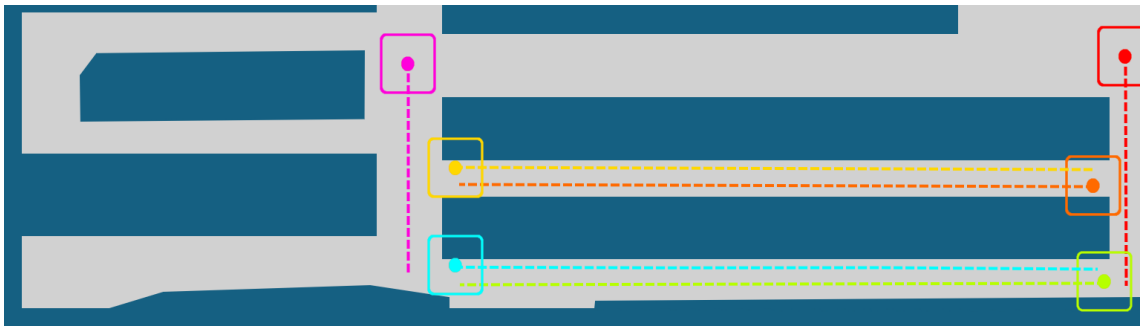


Figur 4.1: Användargränssnitt för den digitala kameran.

Under aktivt träningsläge är ändamålet att skapa syntetisk annoterad träningsdata som sedan kan användas vid träning av ML-modeller. I detta arbete avser begreppet annoterad träningsdata bildpar bestående av en RGB-bild och en segmenteringskarta av samma scen i simuleringen. Därav behöver varje kamera skapa två bilder, en vanlig RGB-bild och en segmenteringskarta, av samma scen under varje renderingsanrop. För att rendera en RGB-bild av simuleringsmiljön användes USceneCaptureComponent2D. Därefter användes UTextureRenderTargetFormat för att skapa en behållare, som renderingen av RGB-bilden sedan placerades i. För att sedan skapa en segmenteringskarta behövdes först diverse klasser, som ska representera objekten simuleringsmiljön, bestämmas. I detta arbete valdes dessa klasser till följande: golv, hinder, människa och robot. Därefter användes USceneCaptureComponent2D och UTextureRenderTargetFormat på liknande sätt för att skapa segmenteringskartan.

En viktig skillnad mellan de båda utförandena var däremot att en efterbehandlingsprocess först behövde appliceras på den renderade bilden, som sedermera blev segmenteringskartan, innan den lades i en Render Target. För att möjliggöra denna efterbehandlingsprocess skapades ett material som bestämmer vilken färg varje klass ska tilldelas. För att åstadkomma detta förses varje meshmodell i simuleringsmiljön med ett ID, beroende på vilken klass objektet tillhör. Sedan använder materialet sig utav logiken, som kan ses i Appendix A.3, vilken utnyttjar dessa ID:n för att tilldela en specifik färg till varje pixel i renderingen. På så sätt skapades en RGB-bild och en segmenteringskarta av samma scen under varje renderingsanrop. Därefter har kameran förmågan att spara detta bildpar lokalt på datorn.

Vid insamlingen av data kan det även vara fördelaktigt att låta kamerainstanserna förflyttas för att öka variationen på data. Detta då en statisk kamera enbart kommer att addera identiska renderingar av samma scen så fort ingen dynamisk karaktär befinner sig inom upptagningsområdet. Huruvida en kamera är statisk eller rörlig bestäms av variabeln Is Moving, se Figur 4.1. Vid ett tillfälle genererades träningsdata med hjälp av sex rörliga kameror. Deras placering och bana illustreras i Figur 4.2.



Figur 4.2: Placering av rörliga kameror vid användande av träningsläge. Prick motsvarar kamerans placering, fyrkant motsvarar grovt det område som avbildas och streckad linje motsvarar kameransfärdväg.

Detta utförande resulterade i en samling data, bestående av totalt 6545 bilder. En fjärdedel av dessa bilder har en låg mättnad för att efterlikna bilder tagna från verkliga kamerorna inuti Volvos Tuves fabrik, se Appendix A.3. Resterande bilder i samlingen har oförändrad mättnad, i syfte att öka variationen på datasamlingen.

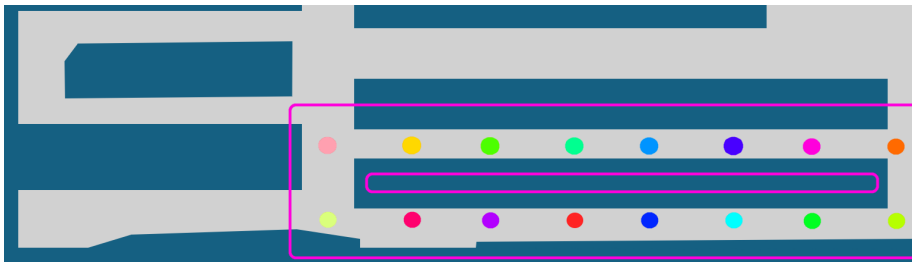
4.1.2 Användning av bilddata för inferens av maskininlärningsmodeller

Under aktivt simuleringsläge är ändamålet istället att enbart rendera RGB-bilder av 3D miljön som sedan kan användas under inferens av färdigtränade ML-modeller för applikationer såsom lokalisering. Därav behöver varje kamera under detta driftläge endast skapa en RGB-bild per renderingsanrop. För att åstadkomma detta användes, likt utförandet för träningsläget, USceneCaptureComponent2D och UTextureRenderTargetFormat. Under inferens av färdigtränade ML-modeller krävs att

simuleringsmiljön på något vis är integrerad med modellerna. För att etablera ett informationsflöde mellan UE5 och ML-modellerna, där de digitalt förvärvade bilderna av den uppbyggda fabriksmiljön fungerar som indata för modellerna, används tillägget TempoROS. TempoROS är ett open source tillägg som erbjuds av Tempo som presenterades i avsnitt 2.4.3. Genom användning av TempoROS kan varje kamera integreras som en separat ROS 2-nod, vilket möjliggör för kommunikation mellan simuleringen i UE5 och externa system, i detta fall maskininlärningsmodellerna.

Att rendera en bild av en scen i UE5 kan dessvärre resultera i en hög beräkningskostnad beroende på ett flertal faktorer såsom: vad för objekt scenen innehåller, bildkvalitet och renderingsfrekvens. Var på de två sistnämnda faktorerna kan anpassas utefter behov, se Figur 4.1. Dessvärre är faktorn som belyser vad för sorts objekt som scenen innehåller ofta okontrollerbart. Summerat över flera kamerainstanser kan detta i värsta fall påverka simuleringshastigheten negativt, vilket i sin tur påverkar det tidigare nämnda informationsflödet. Därav har en positionsspårare, vid namn Tracker, utvecklats vars huvuduppgift är att stänga av kameror när ingen dynamisk karaktär befinner sig i närheten. För att möjliggöra detta behöver först varje kamerainstans och instans av dynamisk karaktär manuellt registreras i positionsspårarens inställningar. Därefter använder spåraren sig utav den inbyggda UE5 funktionen GetActorLocation() för att fastställa varje dynamisk karaktärs världsposition inuti simuleringsmiljön. Denna information används sedan för att beräkna avståndet mellan varje kamerainstans och varje dynamisk karaktär. I detta arbetet har det valts att enbart använda positionsspåraren under simuleringsläget med en uppdateringsfrekvens på 10 Hz, samt att stänga av kamerorna ifall detta avståndet överskrider 600 cm. Dessa värden valdes för att garantera att samtliga dynamiska objekt i simuleringsmiljön alltid fångas upp av en kamera.

Vid inferens av färdigtränade ML-modeller ansågs det inte längre finnas något behov av rörliga kameror utan enbart att kamerornas upptagningsområde täckte hela simuleringsmiljön. Därav användes det totalt sexton statiska kameror vid användning av detta driftläge. Med avsikt att rendera samtliga delar av den spelbara ytan, se Avsnitt 3.5.3, placerades dessa kameror ut med ett mellanrum på 800 cm vilket kan ses nedanför i Figur 4.3.



Figur 4.3: Placering av statiska kameror vid användande av simuleringsläge. Prick motsvarar kamerans placering, fyrkant motsvarar grovt det område som avbildas och streckad linje motsvarar kameransfärdväg.

4.1.3 Utökning och indelning av genererad data

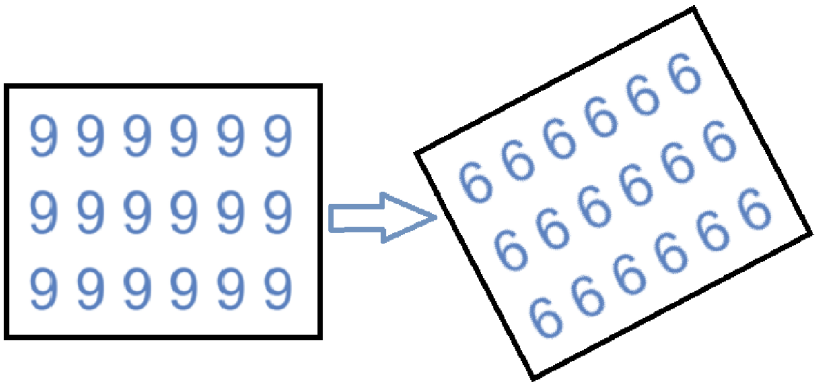
När en mängd annoterad träningsdata har samlats in kan metoden dataökning användas. Dataökning utförs primärt i syfte att öka mängden på samlingen data, då en större samling data kan vara tidskrävande och dyrt att erhålla. Metoden kan även användas för att öka variationen hos den insamlade data, genom att ändra ljusstyrka, rotation och orientering på bilderna, vilket ytterligare utökar den befintliga mängden data [21]. En kombination av dessa sorters bildmanipuleringar syftar till att öka generaliseringen hos maskininlärningsmodellen för annorlunda fabriksmiljöer och öka dess robusthet mot förändringar i Volvo Tuves fabrik.

I detta arbete användes 80 % av den insamlade data, som var totalt 6545 bilder med bildupplösning 960x540, som träningsdata. Denna data utökades offline, det vill säga att datasamlingen utökades en gång för att sedan sparas lokalt på minnet, till totalt en samling av träningsdata på 10000 bilder. På så sätt kunde samma träningsdata användas vid träning av olika segmenteringsmodeller, vilket analyseras i det senare avsnittet 4.2. Resterande 20 % utökades inte utan delades upp i hälften validerings data, som användes under träningen för att övervaka om modellen generaliserar eller överanpassar och hälften användes som testdata för utvärdering av den tränade modellen.

För att utöka träningsdata applicerades affina transformationer på slumpmässigt utvalda bilder, för att införa rotationer och horisontella speglingar, se appendix A.4. Dessutom tillämpades ändringar av ljusstyrka och kontrast, se appendix A.4. Rotation valdes slumpmässigt inom intervallet $[-30°, 30°]$ eftersom bildernas innebörd riskerar annars att förändras vid ett större intervall [22], vilket kan ses i Figur 4.4 där en bild på siffran nio har roterats sådant att den kan uppfattas som siffran sex. Då kamerorna i den verkliga fabriken kan vara vinklade åt olika håll kan horisontell spegling öka maskininlärningsmodellernas invarians mot speglingar [22]. Horisontell spegling utfördes med en sannolikhet på 50 % för varje slumpmässigt vald bild. Därefter sänktes eller höjdes ljusstyrkan och kontrasten slumpmässigt med ett värde på 30 % respektive 20 %. De nämnda värdena valdes med avsikt att försöka spegla de verkliga förhållandena i Volvo Tuves fabrik, där belysningen kan variera mellan olika avdelningar. Dessa värden ansågs vara rimliga då inga extrema ändringar av ljusförhållanden förekommer i fabriksmiljön.

4.1.4 Filter för ovanliga händelser

Volvos fabriksmiljö i Tuve är dynamisk och oförutsedda händelser kan ske, därför undersöks hur de tränade modellerna presterar vid sådana tillfällen. I fabriksmiljön kan det förekomma damm och smuts, det kan hända att kameror skadas eller att ljuskvaliten på kameran förändras. För att efterlikna sådana händelser används tre olika transparenta .png bilder som placeras över två bilder från simuleringsmiljön, i Figur 4.5 visas en av bilderna med de olika filterna. Dessa bilder segmenteras med segmenteringsmodellerna som beskrivs senare, där resultaten kan ses i appendix A.8.



Figur 4.4: En bild som visar hur för mycket rotation av en bild kan ändra innebörden av en bild. Klassindex 9 kan istället tolkas som 6 i detta exempel.

(a) Bild utan filter.

(b) Bild med ljusskiftning.

(c) Bild med repor.

(d) Bild med smuts.

Figur 4.5: Olika filter applicerade på en bild från kamerorna i simuleringsmiljön.

4.2 Jämförelse av segmenteringsmodeller

För att möjliggöra detektering av objekt i simuleringsmiljön av fabriken kan segmenteringsmodeller, maskininlärningsmodeller som använder de tidigare nämnda segmenteringsmetoder för att dela upp insamlad träningsdata, användas. I ett tidigare kandidatarbete [3] användes Pytorchs DeepLabV3_resnet50 [1] för att möjliggöra

[1] `https://docs.pytorch.org/vision/main/models/generated/torchvision.models.segmentation.deeplabv3_resnet50.html`

semantisk segmentering av fotorealistiska digitala representationer av Volvo Tuves fabriksmiljö. Detta utfördes med hjälp av överföringsinlärning. DeepLabv3-resnet50 är en förtränad modell implementerad med Pytorch som är baserad på konvolutionella neurala nätverk (CNN), där ResNet50 extraherar hierarkiska bildrepresentationer medan DeepLabV3 använder dessa representationer för att bestämma vilken klass pixlarna tillhör [23]. I arbetet [3] togs det inte hänsyn till inferenstiden hos modellen, utan enbart förmågan att klassificera. På grund av att detta arbete avser att både klassificera samt lokalisera AMR:er i realtid, motiverar därför det här en undersökning av alternativa modeller. Den verkliga fabriksmiljön som ska digitalt representeras är dessutom komplex då faktorer såsom: likartade objekt, förekomst av smuts och att objekt kan dölja varandra, även kallat ocklusion, är vanligt förekommande. Denna komplexitet av fabriksmiljön gör att global kontextinformation blir viktig för korrekt segmentering, snarare än enbart lokala pixelbaserade egenskaper som CNN baserade segmenteringsmodeller innehar.

Många etablerade segmenteringsmodeller bygger på just CNN-nätverk, men under senare år har transformer-baserade modeller samt kombinationer av CNN och transformer visat förbättrad prestanda i semantisk segmentering, särskilt genom bättre modellering av global kontext [24][25]. Arkitekturen för transformerbaserade segmenteringsmodeller är, likt CNN-nätverk, baserade på djupinlärningsarkitektur. En distinkt skillnad som urskiljer transformers ifrån CNN-nätverk är så kalad self-attention. Detta är en kritisk mekanism i transformerarkitekturen som tillåter modellering av global kontext [26]. Därmed rättfärdigas en undersökning av transformer-baserade modeller för segmentering av den uppbyggda simuleringsmiljön. I detta arbete kommer därför en transformer-baserad modell, SegFormer-B0, att tränas för att sedan användas till semantisk segmentering.

Vidare undersöks även om instanssegmentering kan vara en mer lämplig metod än semantisk segmentering för den aktuella uppgiften. Till skillnad från semantisk segmentering möjliggör instanssegmentering särskiljning av enskilda objekt av samma klass. Särskiljning av objekt är centralt för styrning av AMR. Med avsikt att undersöka potentialen hos instanssegmentering för delproblemets ändamål används därmed modellen YOLOv8-Seg Nano.

För att, de tidigare nämnda, modellerna ska kunna lokalisera objekten i simuleringsmiljön behöver de först tränas. För att effektivisera denna del av arbetsprocessen kommer träningsmetoden överföringsinlärning att användas, vilket beskrivs i följande avsnitt. Därefter diskuteras och jämförs de två segmenteringsmodellerna: SegFormer-B0 och YOLOv8-Seg Nano.

4.2.1 Överföringsinlärning

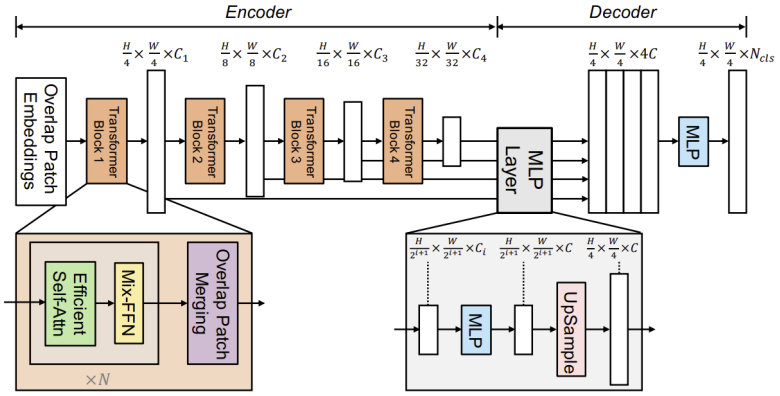
Överföringsinlärning [27] är en träningsmetod inom maskininlärning som är användbar vid storleksbegränsade dataset. I överföringsinlärning använder man en förtränad modell som blivit tränad på ett mycket större dataset men på en liknande domän för att undvika träning av en modell från grunden. Eftersom den förtränade mo-

dellen redan har lärt sig kontext från bilder på liknande objekt och miljöer behövs endast vissa lager av modellen uppdateras, vilket ofta benämns som finjustering. Vanligtvis uppdateras endast de sista lagren, det vill säga klassificeraren, med nya klasser medan de resterande lagren är orörda.

Då storleken på arbetets dataset inte kan säkerställas tillämpas överföringsinlärning i ett förebyggande syfte. Vid genomförandet av denna träningsmetod kommer först det sista lagret i modellerna att uppdateras med nya klasser som beskriver den uppbyggda simuleringsmiljön. Därefter kommer samtliga lager utom det sista lagret att vara låsta under träning. På så sätt utnyttjas den kontext som modellerna redan har lärt sig från större dataset, vilket är särskilt viktigt då den genererade samlingen av data från UE5 är begränsad i storlek jämfört med dataseten modellerna är förtränade på.

4.2.2 SegFormer

SegFormer [24] är en transformer-baserad modell specifikt anpassad för semantisk segmentering. Till skillnad från generellt långsamma transformer-arkitekturer [26], speciellt i samband med högre bildupplösning, använder SegFormer en lättviktig flerlagers perceptron (MLP) avkodare. Kodarens tidiga lager är av högre upplösning och kan därav fånga upp lokala mönster som kanter och hörn, medan de högre lagren är av lägre upplösning vilket kopplar ihop större delar av bilden för global kontext. MLP-avkodaren utnyttjar sedan både den lokala och globala kontexten från kodarens lägre och högre lager för att avgöra vilken klass varje pixel tillhör [24]. Se Figur 4.6 för visuell representation av SegFormer arkitektur.



Figur 4.6: SegFormer arkitektur, där uppdelningen av avkodare som benämns som avkodare och kodaren som benämns som kodare visualiseras. Bilden är tagen från [24] och har fått tillåtelse att användas.

I originaluppsättningen av SegFormer-modellerna [24] används olika kodare beroende på vilken modell som valts. Här omfattas varianterna av kodare av intervallet MiT-B0-MiT-B5. MiT-B0, som används i SegFormer-B0, är den lättaste kodaren eftersom den består av färre parametrar vilket leder till snabbast inferenshastighet bland de övriga varianterna av kodare [24]. Därtill, en fördel som beskrivs om

SegFormer är att den kan ta in bilder av olika upplösningar under inferens [24], vilket kan vara användbart om bilder från verkligheten är av en annan upplösning än det som modellen är tränad på. För att kunna uppnå realtids segmentering av simuleringsmiljön kommer därav modellvarianten SegFormer-B0 att användas i detta arbete.

Träning

Till träning användes en SegFormer-B0 modell som var förtränad på datasamlingen ADE20K [28]. Eftersom ADE20K består av bilder som representerar bland annat människor och bilar samt amorfa områden såsom vägg och väg [29] ansågs domänen för denna datasamling vara godtycklig mot domänen för den insamlade träningsdata, som beskrivs i avsnitt 4.1.1. Därav betraktades den valda förtränade modellen som lämplig.

Inför träningen behövdes diverse hyperparametrar samt designval utvärderas och justeras. På så sätt kunde träningsprocessen styras för att uppnå önskat resultat. Följande parametrar beskrivs nedan.

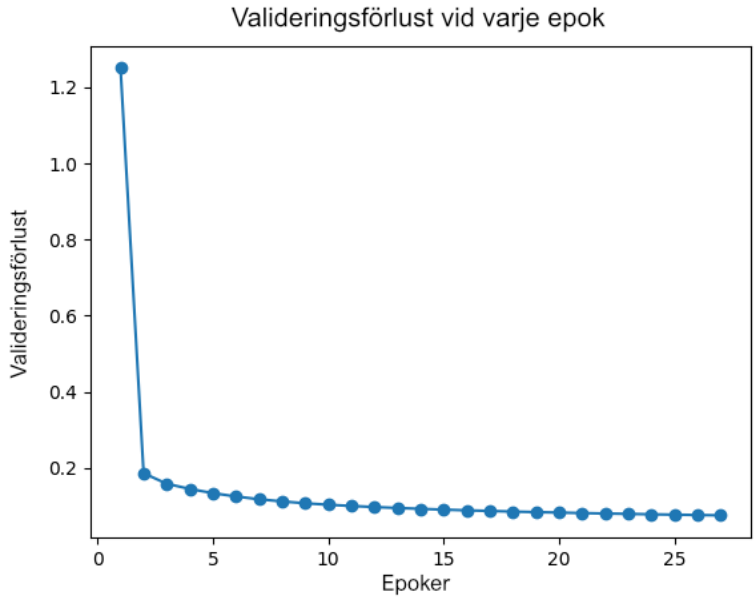
- CrossEntropyLoss användes i träningen då det används och har visat bra resultat i många segmenteringsmodeller [30][31]. Det väljs automatiskt av den färdigbyggda modellklassen SegformerForSemanticSegmentation från Hugging Face [2].
- Adam optimeringsalgoritmen tillämpades för att uppdatera modellens vikter under träning eftersom den anpassar modellens inlärningshastighet [32] och har visat på snabb och stabil konvergens [33].
- Inlärningshastighet på optimeringslagoritmen sattes till 3×10^{-5}, för att bevara förtränade vikter och därmed få bättre träning av modellen [34].
- Träningen pågick i maximalt 80 epoker men EarlyStopping-metoden[3] från PyTorch tillämpades för att stoppa träningen när valideringsförlusten inte förbättrades med mer än 0.005 efter 5 epoker för att undvika överanpassning till träningsdata.
- För att spara modellen som uppnådde lägst valideringsförlust under träningen användes metoden ModelCheckpoint[4] för att automatiskt spara den bästa modellen vid träningens slut och inte enbart den senast tränade modellen.

Under träningen var hela kodaren, förutom det sista lagret block 4, se Figur 4.6, låst för att tillämpa överföringsinlärning. Det var alltså endast parametrarna i block 4 samt avkodaren som uppdaterades under träningen. Detta beslut togs då klasserna i ADE20K inte överensstämde exakt med de klasser som beskriver den simulerade fabriksmiljön.
För att se hur valideringsförlusten förändrades vid varje epok skapades en graf, se Figur 4.7. I grafen som visualiseras går det att avläsa hur träningen slutade efter

[2] `https://huggingface.co/docs/transformers/v5.8.0/en/model_doc/segformer#transformers.SegformerForSemanticSegmentation`

[3] `https://docs.pytorch.org/ignite/generated/ignite.handlers.early_stopping.EarlyStopping.html`

[4] https://lightning.ai/docs/pytorch/stable/api/lightning.pytorch.callbacks.ModelCheckpoint.html

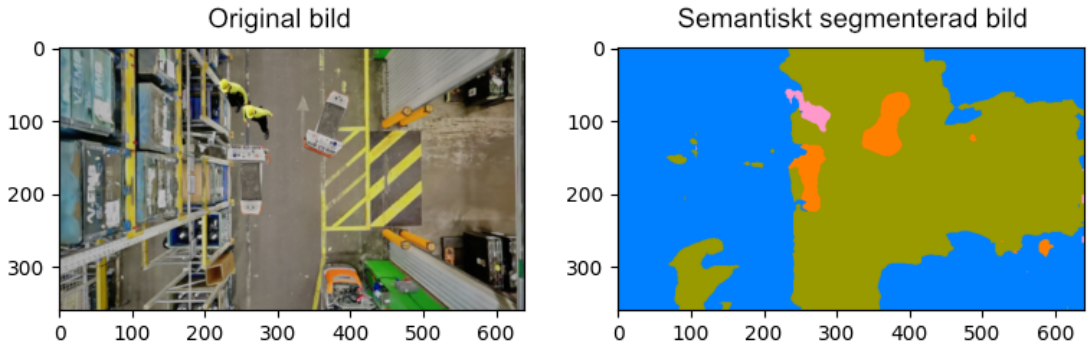


Figur 4.7: Jämförelse mellan ökande antal epoker och utvecklingen av validerings förlusten.

26 epoker och att förlusten, vid den epoken, hade konvergerat. Från grafen går det även att utläsa hur förlusten sjönk snabbt i början för att sedan stabilisera sig vilket tyder på en stabil träning.

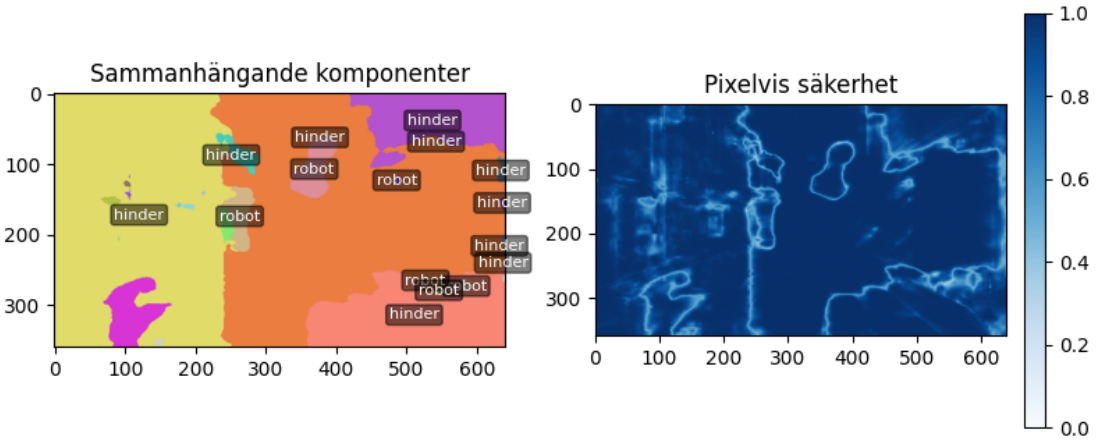
Inferens

Inferens innebär att den förtränade modellen, i detta fall SegFormer-B0, segmenterar en bild från testdata, som beskrevs i avsnitt 4.2.2. Först skalas bilden om till lägre upplösning och pixelvärdena normaliseras, från färgskalan 0-255 till 0-1. Därefter genererar modellen råpoäng för varje klass, ofta kallade logits, vilka baseras på vikterna i den förtränade modellen. Dessa poäng skalas sedan upp till bildens ursprungliga upplösning, så att en klass kan bestämmas för varje pixel. Klassen med högst poäng blir pixelns klassetikett. Eftersom varje klass är kopplat till ett index kan varje index på så sätt tilldelas en RGB-färg för visualisering som i Figur 4.8.
Tidigare nämndes att instanser kan användas till styrning av AMR:er, för att erhålla instanser tolkas den semantiskt segmenterade bilden. För att två objekt av samma klass ska identifieras som två instanser behöver dessa separeras med pixlar från en annan klass. För att tydligt visualisera varje instans i bilden kan en etikett med dess klass visas. Genom att beräkna medelvärdet av instansernas x- och y-koordinater kan instans centroiderna bestämmas vilket används till att sätta ut klass etiketter för visualisering av instanssegmentering men också för att i senare skede av arbetet kunna beräkna avståndet av objekt för reglering av robotar. Därtill gjordes även en karta för att visualisera den tränade modellens pixelvisa säkerhet som representeras med en skala 0-1 och motsvarande vit-blå färgskala. För att beräkna den pixelvisa säkerheten konverteras logits, som nämnts tidigare, till en skala 0-1 så att sanno-



Figur 4.8: I bilden till vänster visas originalbilden från kamerorna i den simulerade fabriken. Till höger visas den semantiskt segmenterade original bilden segmenterad med den tränade SegFormer-modellen, där grönt representerar golv, orange representerar robot, rosa representerar människa och blå är hinder.

likheter för varje klass av en pixel summeras till 1. De högsta logits för varje pixel väljs ut som värden för den pixelvisa säkerheten och på så sätt kan det visualiseras i en karta. Instansindelningen och kartan med den pixelvisa säkerheten visas i Figur 4.9.



Figur 4.9: I bilden till vänster visas sammanhängade komponenter, dvs. instanserna i den semantiskt segmenterade bilden. Till höger visas en karta på modellens pixelvisa säkerhet, där blå visar att modellen är väldigt säker och vitt visar att modellen är osäker på den pixelvisa klassindelningen.

För att beräkna inferenstiden av den tränade modellen användes Python funktionen `perf_counter()` från modulen time i standardbiblioteket. Först behövde modellen varmstartas, därav kördes segmenteringen en gång innan tidtagaren startades. Därefter beräknades ett medelvärde av 13 slumpmässigt utvalda bilders inferenstid för att få en sammanfattande och stabil beskrivning av inferenstiden. Mätvärdena som användes för att åstadkomma detta kan ses i tabell 4.2 och den genomsnittliga inferenstiden beräknades vara 0,0224 s per bild.

Tabell 4.1: Inferenstider för den tränade SegFormer-B0 modellen per bild.

Mätning	Inferenstid (s)
1	0,0219
2	0,0224
3	0,0225
4	0,0238
5	0,0222
6	0,0232
7	0,0220
8	0,0222
9	0,0219
10	0,0223
11	0,0225
12	0,0223
13	0,0222

4.2.3 YOLO

YOLOv8-Seg är en segmenteringsmodell baserad på CNN-nätverk som använder kodaren EfficientNet-B4 [35] utvecklad av Ultralytics för instanssegmentering. EfficientNet-B4 består av 71 lager och 19 miljoner parametrar, den genererar fem funktionskartor P3-P7, funktionskartor är vad filter i konvolutionella nätverk returnerar när de matas in med en bild eller resultatet från föregående filter. På engelska kallas de för feature maps. P3-P7 har olika upplösningar och storlekar där P3 har högst upplösning och P7 har lägst. YOLOv8-Seg använder NAS-FPN-Cell [11], ett 6 lager nätverk som gör prediktioner om vad bilden innehåller, och matas in med dessa 5 funktionskartor. Därefter returneras fem nya feature maps av samma storlek som sedan kan användas till objektdetektering.

YOLOv8-Seg har fem olika storleksvarianter där storleken leder till ökad precision till bekostnad av lägre inferenshastighet. Av högst intresse är den minsta modellen YOLOv8-Seg Nano för att uppnå högre inferenshastighet.

Träning

Träningen utfördes med YOLOv8-Seg Nano förtränad på datasetet COCO [36]. Datasetet inkluderar bilder på människor men inte områden såsom väggar och golv. Eftersom väggar och golv ansågs ha simpla geometrier och liten variation medans människor var en så viktig del för segmenteringen ansågs datasetet fortfarande vara lämpligt.

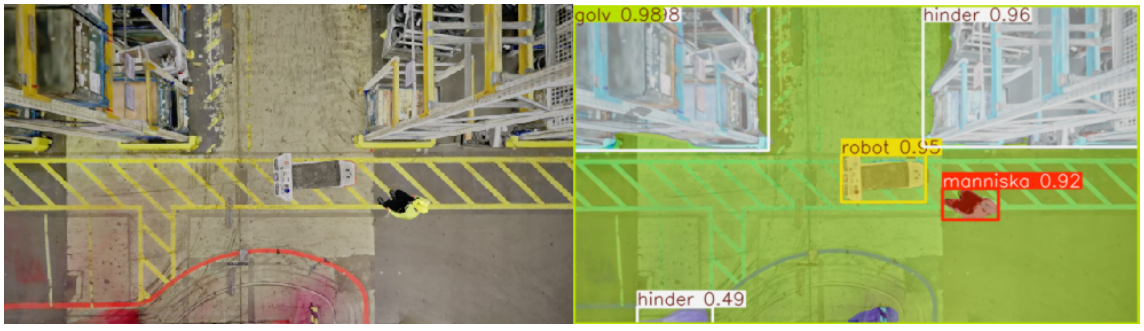
Innan träningen började behövde segmenteringskartor konverteras om till textfiler som innehåller varje pixels klass-id. Inställningarna inför träning justerades till klass antalet till fyra klasser, samt ändrades klasserna till golv, robot, människa och hinder. Inför träning av den förtränade modellen YOLOv8-Seg Nano behövde även

hyperparametrar justeras enligt följande:

- Den förlustfunktion som användes var Focal Loss, eftersom det är en standard funktion [11] som används vid träning av YOLO-modeller. Focal Loss funktionen balanserar inlärningen så att objekt som är viktiga men inte förekommer ofta i träningsdata, såsom robotar, väger tyngre än övriga frekvent förekommande objekt.
- Inlärningshastigheten bestäms till en start inlärningshastighet på 0,01 samt en slut inlärningshastighet på 0,0001, vilka är standardvärden för YOLO-modeller.
- Träningen pågick i 40 epoker. Men parametern Patience ställdes in på 10 epoker, vilket innebär att om modellen inte förbättrades på mer än 10 epoker, så avslutades träningen. Efter träningen sparades en modell med bäst resultat samt den senast tränade modellen.
- Parametern Workers, bestämmer hur många CPU-processer som används samtidigt och sattes till 2 för att minska träningstiden.
- Mosaic sattes till 0 för att stänga av ytterligare utökning av träningsdata under träningsprocessen.
- Överföringsinlärning tillämpades genom att parametern freeze sattes till 10 så att hela kodaren var låst under träningen[5].

Inferens

Under inferensen tilldelas varje pixel en klass, där varje objekt tilldelas därefter en instans-ID. Under inferensen beräknas även lokalisering av varje instans med hjälp av centroider som beskrevs tidigare i avsnitt 4.2.2. Tillsist placeras de segmenterade maskarna på originalbilden, ofta kallad overlay, för att skapa en segmenterad bild. Därtill visualiseras objektrutor och klassetiketter med modellsäkerheten för varje instans, vilket visas i Figur 4.10.



Figur 4.10: I bilden till vänster visas originalbilden från kamerorna i den simulerade fabriken. Till höger visas den segmenterade bilden ovanpå original bilden. Klasserna representera som följande: vitt för hinder, gult för robot, rött för människa och grönt för golv

[5]https://docs.ultralytics.com/guides/custom-trainer

Tabell 4.2: Inferenstider för den tränade YOLOv8-Seg Nano modellen per bild.

Mätning	Inferenstid (s)
1	0,0593
2	0,0516
3	0,058
4	0,0593
5	0,0588
6	0,0576
7	0,0579
8	0,0596
9	0,0579
10	0,0581
11	0,059
12	0,0574
13	0,0585

På samma sätt som inferenstiden beräknades för SegFormer-B0, beräknades den genomsnittliga inferenstiden per bild till 0,0579 s för den tränade YOLOv8-Seg Nano modellen baserat på de mätvärden som presenteras i Tabell 4.2.

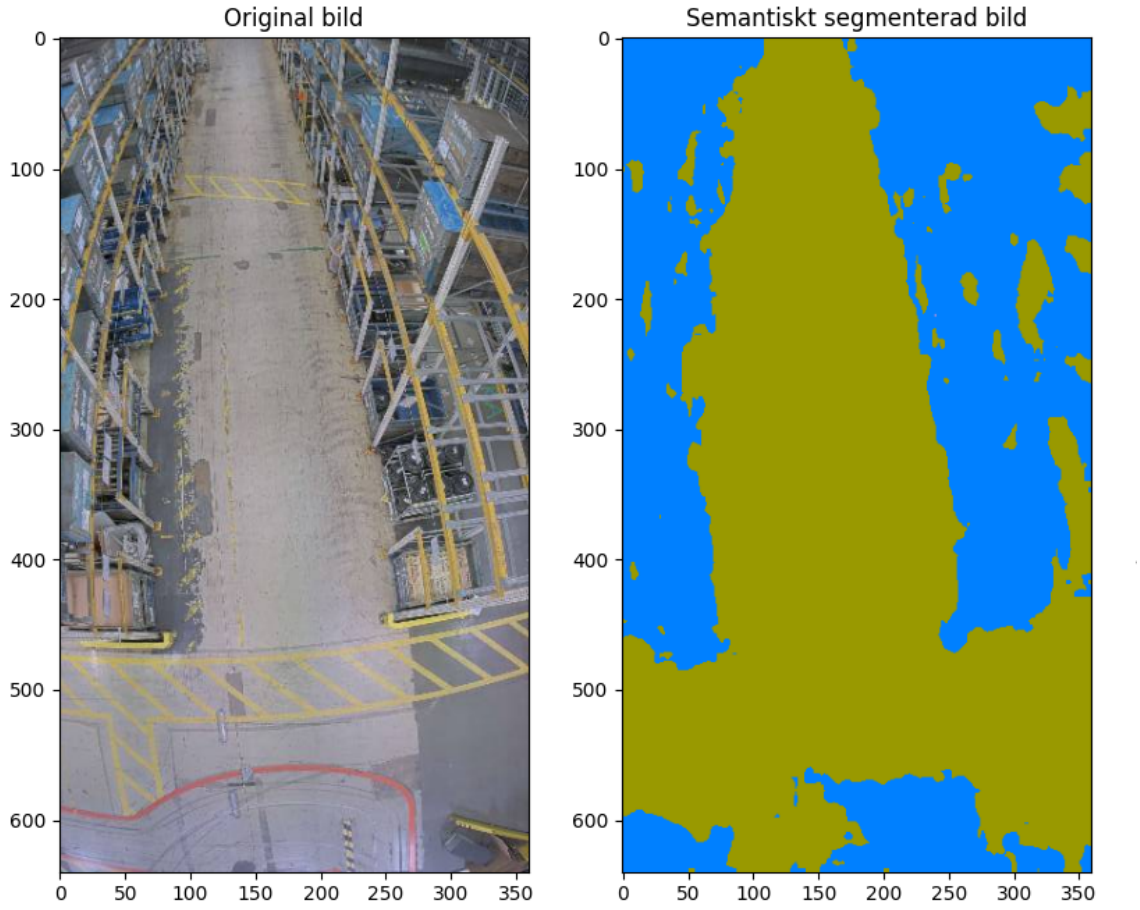
4.2.4 Val av segmenteringsmodell

För att utvärdera och jämföra de två segmenteringsmodellerna SegFormer-B0 och YOLOv8-Seg Nano bedöms flera olika aspekter. Aspekterna som bedöms är hur väl de tränade modellerna presterar på bilder tagna från den riktiga Volvo fabriken i Tuve, modellerna bedöms utifrån prestationsmåtten mIoU, IoU samt mAP. Utöver det bedöms även inferenshastigheten och kvalitativt hur robusta modellerna är mot ovanliga händelser med kameran i detta avsnitt. För att se alla segmenterade bilder se Appendix A.5 och A.6.

Sim-to-Real gapet

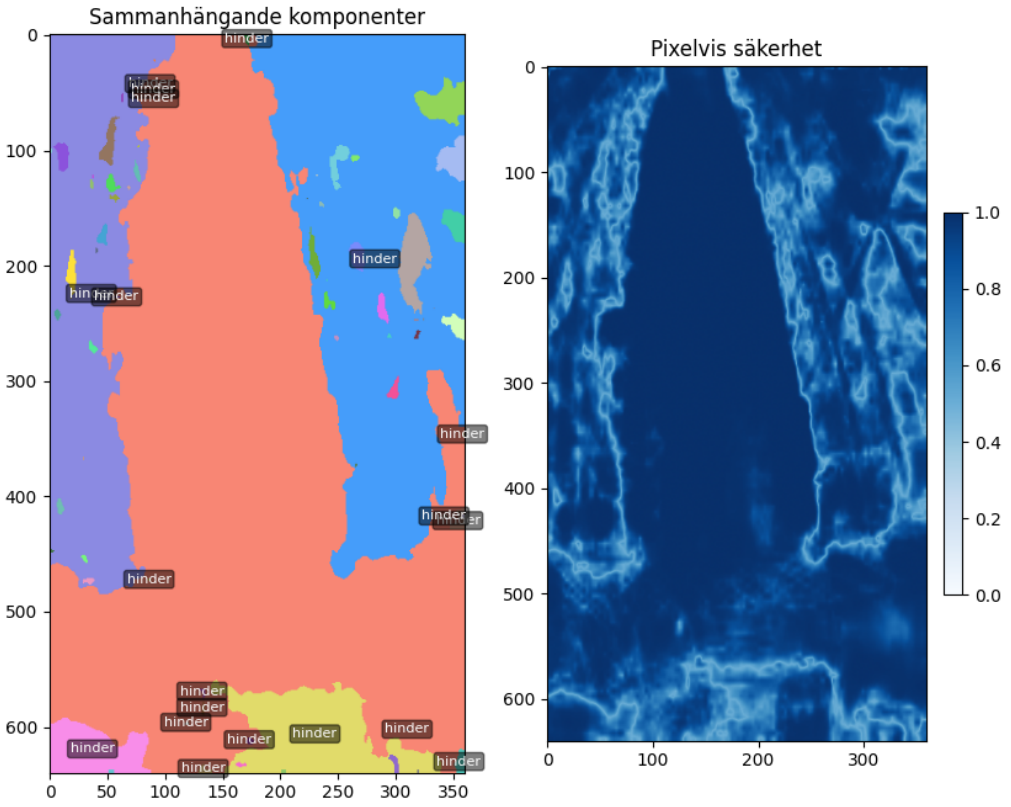
En viktig del av utvärderingen av de tränade segmenterings modellerna är att utvärdera Sim-to-Real gapet. Simuleringen beskriver inte verkligheten perfekt och därför kommer det att finnas skillnader mellan verkligheten och hur det ser ut i simuleringen. Ett lägre Sim-to-Real gap eftersträvas [37].

Eftersom modellerna enbart tränades på syntetiska bilder kan det finnas en skillnad mellan dem och bilderna som tas från kamerorna i den riktiga fabriken. Det är därför viktigt att testa hur modellen presterar på de riktiga bilderna. För att utvärdera Sim-to-Real gapet används 4 bilder från tak-monterade kameror i Tuve fabriken, för att testa hur bra de tränade modellerna presterar på bilder från verkligheten. Bilder från verkligheten segmenterades med de tränade YOLOv8-Seg Nano och SegFormer-B0 modellerna, likt Figur 4.11,4.12 och 4.13, för att se resten av bilderna se Appendix A.7.



Figur 4.11: Original bild till vänster och den semantiska segmenteringen av den till höger, som gjordes med hjälp av den tränade SegFormer-B0 modellen. Gröna partier representerar klassen golv och de blåa partierna representerar klassen hinder.

Från Figur 4.11, 4.12 och 4.13 kan man bedöma att båda modellerna kan identifiera klasserna golv och hinder. Skillnaden är att bilden segmenterad med YOLOv8-Seg Nano missar största delen av golvet och gissar fel, dessutom segmenteras inte hela bilden, hälften av bilden förblir icke segmenterad. Bilden segmenterad med SegFormer blir däremot helt segmenterad, men det förekommer fel som att vissa delar av golvet missklassificeras som hinder. Nackdelen med att beräkna instanser med den tränade SegFormer modellen är ganska tydlig i figur 4.12, där man kan se att även små obetydliga separationer mellan pixel-klasser skapar flera instanser, alltså att ett hinder kan då istället tolkas som flera hinder, vilket är missvisande. Då är YOLOv8-Seg Nano bättre för ändamålet att få ut instanser, i figur 4.12 blir hinder två instanser. Om man jämför SegFormers pixelvisa säkerhet mot YOLOv8-Seg Nanos instans säkerhet kan man från SegFormer-modellen tolka att den är sämre på kanter och hörn och att den är mindre säker på hinder än golvet. YOLOv8-Seg Nano är däremot relativt säker på hinder, men det skiljer sig mellan de två olika hinder, samtidigt är den väldigt osäker på golv instansen vilket är relativt rimligt



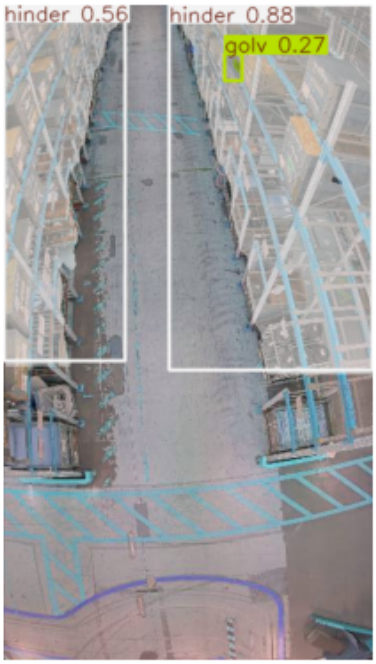
Figur 4.12: Instanser som kunde produceras visas i sammanhängande komponenter till vänster. Till höger visas en karta med den pixelvisa säkerheten på den semantiska segmenteringen av en riktig bild från Volvo Tuves fabrik.

då den gissar fel. I övriga bilder som finns i Appendix A.7 gissar modellerna relativt lika men YOLOv8-Seg Nano missar att segmentera vissa delar av bilden i A.23.

De skillnader som uppstår mellan syntetiska bilder och bilder tagna från Volvo fabriken kan bero på att kameravinklarna inte riktigt överensstämmer med det modellerna är tränade på, dessutom är linsen på kamerorna i Volvo fabriken mer utvidgade än i träningsdatan, samt att bildkvaliteten skiljer sig åt. Men överlag presterar båda modellerna relativt lika, även om YOLOv8-Seg Nano ibland missar att segmentera vissa delar av bilderna. I segmentering med båda modellerna förekommer fel, men det är väntat då modellerna är endast tränade på syntetisk träningsdata. Dessutom är det bara 4 tillgängliga bilder från verkligheten som testas på modellerna, vilket är inte rättvisande, och är en begränsning, och för att ge en god bedömning bör fler bilder med fler situationer utvärderas.

Prestationsmått

Måttet Intersection over Union [6] som ofta förkortas till IoU beräknar förhållandet mellan sant positiva (TP) och summan av falskt positiva (FP), falsk negativa (FN) och sant positiva (TP) klassetiketter som i ekvation 4.1. IoU mäts på en skala 0 till



Figur 4.13: Instanssegmentering med YOLOv8-Seg Nano på en bild tagen från Volvo Tuves fabrik. Det vita representerar hinder och det gröna representerar golv.

1, där 1 är det högsta värdet en modell kan få.

$$IoU = \frac{TP}{FP + FN + TP} \tag{4.1}$$

Mean IoU som ofta förkortas till mIoU används för att beräkna ett medelvärde av varje klass IoU, se ekvation 4.2. Detta mått är etablerat i många semantisk segmenterings problem [6] och kommer därför att användas för att bedöma de olika modellerna.

$$mIoU = \sum_{j=1}^{k} \frac{TP_{jj}}{FP_{ji} + FN_{jj} + TP_{ij}} \tag{4.2}$$

Där TP_{jj} innebär alla pixlar som är klassen j och som har bedömts som j, FP_{ji} innebär alla pixlar som är klassen j men bedömts som klass i, FN_{jj} är antalet pixlar som inte är klassen j men som bedömts vara klassen j.

Eftersom YOLOv8-Seg Nano är en instanssegmenterings modell är det missvisande att bedöma modellens noggrannhet med IoU eller mIoU, eftersom IoU och mIoU ignorerar instansindelningen som YOLOv8-Seg Nano gör vilket kan resultera i ett lägre mätvärde. Måtten bedömer endast den pixelvisa noggrannheten. I Tabell 4.3 kan det observeras att YOLOv8-Seg Nano har generellt lägre IoU per klass än SegFormer-B0.

Däremot om man beaktar det som tidigare nämnts är YOLOv8-Seg Nanos värden fortfarande höga och behöver inte innebära att den är sämre än SegFormer-B0. Det kan också bedömas att båda modellerna har väldigt låg IoU värde på klassen människa, vilket kan bero på att människor inte förekommer så ofta i träningsdata, eller att människorna inte skiljer sig så mycket från resten av miljön i bilderna. En skillnad mellan YOLOv8-Seg Nano och SegFormer-B0 är att SegFormer-modellen gissar sämre på klassen robot, även när man jämför med hur modellen själv bedömer klasserna hinder och golv. Det kan bero på att AMR:er i bilderna hade en matta som liknade färgen på golvet. Detta tyder på att modellen tränad med SegFormer kan vara sämre på att identifiera objekt som liknar varandra, jämfört med YOLOv8-Seg Nano.

Tabell 4.3: Jämförelse mellan segmenteringsmodellers IoU per klass.

	IoU med SegFormer-B0	**IoU med YOLOv8-Seg Nano**
Människa	0,184	0,009
Robot	0,616	0,757
Hinder	0,924	0,768
Golv	0,950	0,875

Ur Tabell 4.4 kan det bedömas att mIoU för YOLOv8-Seg Nano är lägre än mIoU för SegFormer-B0, vilket kan bero på att IoU för klassen människa var mycket lägre än för resterande objekt i båda modellerna men särskilt för YOLO-modellen. Överlag har båda modellerna ganska hög mIoU.

Tabell 4.4: Jämförelse av mIoU, FPS och medel inferenstid mellan segmenterings modeller.

Modell	**Mean IoU**	**FPS**	**Medel inferenstid (ms)**
YOLOv8-Seg Nano	0,561	17,26	57,90
SegFormer	0,774	44,82	22,30

För att utvärdera YOLO-modellen mer noggrant mättes även mean Average Precision 50 (mAP50) och mean Average Precision 50-95 (mAP50-95) som är etablerade för bedömning av instanssegmenterings modeller [38]. Måttet Average Precision bedömer precisionen mellan de segmenterade maskarna och segmenteringskartorna som används som ground-truth, för varje klass [39][40]. På så sätt är mean Average Precision (mAP) medelvärdet av Average Precision över alla klasser. Mean AP50 innebär att man mäter mAP när tröskeln för Intersection over Union (IoU) är 50 % medan mAP50-95 innebär att man mäter mAP när tröskeln för IoU är 50 % till 95 % [40]. Mean AP testas över olika IoU trösklar för att ge en mer noggrann bedömning av den tränade modellen [41].

Mean Average Precision 50 och 50-95 med den tränade YOLOv8-Seg Nano blev enligt Tabell 4.5. När man jämför dessa värden med YOLOv8-Seg Nano testad på COCOs dataset, där mAP50 är 36,7 och mAP50-95 är 30,5[6], kan slutsatsen

[6] `https://docs.ultralytics.com/models/yolov8`

dras att både mAP50 och mAP50-95 för den tränade modellen är mycket högre. Eftersom datasetet COCO har 80 klasser och den tränade modellen är tränad på endast fyra klasser är det rimligt att värdena på den tränade modellen är högre än den förtränade modellen. Mean AP50 på 83,8 % visar att den tränade modellen segmenterar och detekterar objekt rätt, och mAP50-95 på 61,9 % visar även att modellen kan producera korrekta segmenterings maskar.

Tabell 4.5: Modell tränad med YoloV8-seg Nano prestanda mätt i mask mAP50 och mask mAP50-95 på testdata.

Mått	**Värde**
mAP50	0,838
mAP50-95	0,619

Inferenstid

Vid lokalisering av objekt i simuleringsmiljön kan det vara fördelaktigt att ha låg inferenstid, därför utvärderas modellernas inferenshastighet.

Ur Tabell 4.4 kan det bedömas att YOLOv8-Seg Nano har en inferenstid på 57,9 ms, medan SegFormer-B0 har en hastighet på 22,3 ms. SegFormer har mycket högre inferenshastighet i denna utvärdering än YOLOv8-Seg Nano, vilket är bra för lokalisering av objekt i simuleringsmiljön.

Ovanliga händelser

Utifrån figurerna i Appendix A.8 kan man visuellt tyda att YOLO-modellen segmenterar ungefär lika bra med filter som utan. SegFormer-modellen är däremot sämre på detta, då den missar att segmentera delar av roboten och missklassificerar golvet som hinder. Detta tyder på att modellen tränad med YOLOv8-Seg Nano kan vara bättre på att segmentera ovanliga bilder. För en bättre bedömning bör däremot fler situationer, fler bilder och förhållanden testas.

Slutsats

Båda modellerna SegFormer-B0 och YOLOv8-Seg Nano har visat bra resultat. SegFormer-B0 har bra mätvärden, relativt korrekta segmenterade bilder och har visat på snabbare inferenshastighet än YOLOv8-Seg Nano. Baserat på resultat visar SegFormer-B0 på bättre värden och kan vara ett bra val. Däremot finns det en kritisk skillnad mellan den tränade SegFormer-B0 modellen och YOLOv8-Seg Nano modellen, och det är att den ena utför semantisk segmentering medan den andra utför instanssegmentering. Arbetet har visat på att det är möjligt att skapa instanser baserat på de semantiskt segmenterade bilderna som skapas under inferens med SegFormer, men dessa instanser är inte helt korrekta på grund av att segmenteringen inte är perfekt. YOLO-modellen däremot producerar mer noggranna instanser direkt vilket gör det mer smidigt när det kommer till reglering av AMR:er som beskrivs i nästa kapitel. Under reglering av roboten är det viktigt att kunna separera mellan olika objekt

för att kunna beräkna dess centroider för objekt lokalisering, vilket blir svårare med SegFormer-modellen. Därtill visade det sig att YOLOv8-Seg Nano var bättre på att rätt segmentera bilder vid oförutsedda kamera händelser vilket tyder på modellens tillförlitlighet och robusthet vid ovanliga situationer, vilket är en fördel i en dynamisk fabriksmiljö. På grund av dessa skäl görs beslutet att använda den tränade YOLOv8-Seg Nano modellen vidare i arbetet.

4.3 Utvärdering

Detta kapitel utvärderar datagenereringen och behandling, tillämpningen av överföringsinlärning samt den valda segmenteringsmodellen.

Beslutet att undersöka både semantisk och instanssegmentering gav värdefull insikt om dess fördelar och nackdelar i att lösa problemen: objektigenkänning, lokalisering och dess inverkan på vidare arbete med reglering av autonoma mobila robotar. Semantisk segmentering kunde användas till att identifiera objektklasser, samtidigt som individuella instanser i vissa fall kunde extraheras genom sammanhängande komponenter från semantiska segmenterings bilder. Men trots SegFormer-modellens höga mätvärden konstaterades det att instanserna som extraherades från segmenterings bilderna inte var lika noggranna som med en riktig instanssegmenterings modell. Baserat på de resultaten beslutades det därför att instanssegmentering var mycket mer användbart för framtida reglering av autonoma mobila robotar. Detta eftersom individuella instanser möjliggör beräkning av avstånd mellan objekten samtidigt som varje objekt av samma klass kan särskiljas.

Baserat på ovanstående bedömdes att den tränade YOLOv8-Seg Nano modellen var mer passande än SegFormer-B0 modellen för att tackla senare problem. Modellen var även snabb och resultaten tyder på att den kan användas för segmentering i realtid, därtill segmenterade den tillräckligt bra för att vara YOLO:s minsta modell. Vid behov av noggrannare segmentering hade därför en undersökning av större modeller rekommenderats.

Däremot en begränsning med instanssegmenteringen som identifierades sent i arbetet var att träningsdata som användes till instanssegmentering och semantisk segmentering var identisk, det vill säga att samma segmenteringskartor användes till båda modellerna. Detta är problematiskt eftersom segmenteringskartor för semantisk och instans segmentering skiljer sig. Trots begränsningen kunde modellen identifiera olika objekt som instanser, vilket kan bero på att instanserna i bilderna förekom utan överlapp och därför kunde särskiljas.

När det kommer till datagenerering och behandling kunde UE5 producera en stor mängd annoterad data. En styrka med att använda UE5 är att färgerna på segmenteringskartorna i den annoterade data kunde justeras, vilket tyder på att segmenteringskartor för instanssegmentering hade kunnat genereras. Filformaten och storleken på den genererade data kunde också justeras vilket kan vara användbart vid träning av en annorlunda datasamling. Dessutom kunde den insamlade bilddata

lätt ordnas i olika fil- och mappstrukturer vilket underlättar processen vid träning av segmenteringsmodeller.

Den insamlade träningsdata kunde utökas med olika transformationer och justeringar. Segmenteringsmodellernas resultat tyder på att utökningen av datasamlingen inte påverkade träningen negativt. En större datasamling hade dock varit fördelaktig för att öka segmenteringsmodellens robusthet mot variation i data..

Dessutom testades den tränade modellen på bilder tagna från den riktiga Volvo fabriken i Tuve för att analysera om syntetisk träningsdata är tillräckligt bra för att modellen ska kunna känna igen objekt i verkliga bilder. I de bilder som prövades kunde modellen segmentera objekt, men detta testades på enbart fyra verkliga bilder, och för att kunna ge en rättvis bedömning bör modellen testas på ett mycket större datasamling av verkliga bilder.

Sammanfattningsvis visar resultaten att det är möjligt att generera stora mängder träningsdata med hjälp av UE5. Den genererade datasamlingen från UE5 kunde användas till träning av segmenteringsmodeller. Segmenteringsmodellerna visade förmåga på att kunna identifiera objektklasser både på simulerade och verkliga miljöer, vilket tyder på att den syntetiska träningsdatan kan användas till generalisering till viss grad även om viss Sim-to-Real gap existerar. Till sist bedömdes instanssegmenterings modellen YOLOv8-Seg Nano vara mer passande för att lokalisera objekt i simuleringsmiljön.

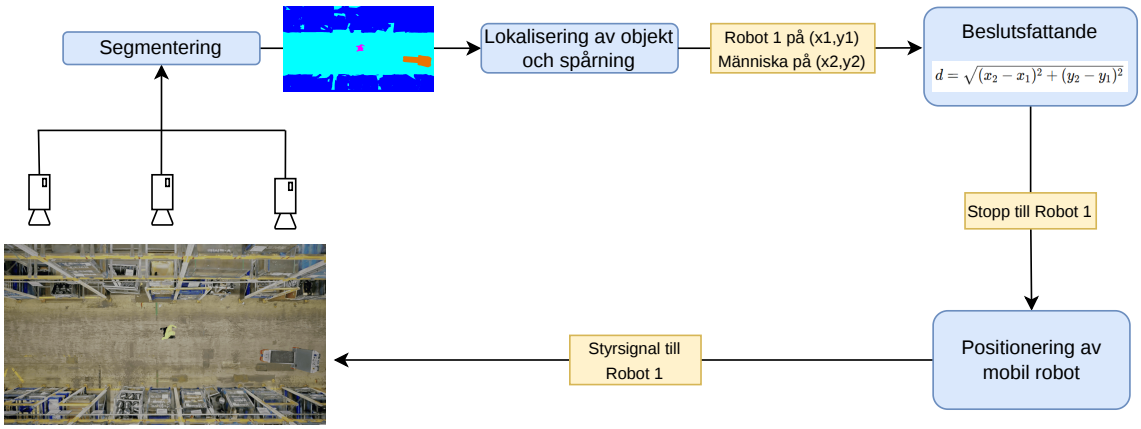
5

Styrning av mobila robotar

Tidigare kapitel 3 och 4 har beskrivit hur simuleringsmiljön har byggts upp och hur perception har använts för att lokalisera simulerade objekt. Följande kapitel bygger vidare på dessa resultat och behandlar hur data för de lokaliserade objekten kan användas för att styra AMR:erna.

När lokaliseringen av simuleringsobjekten har genomförts baserat på bilddata återstår att behandla denna information och omvandla den till styrsignaler som kan användas för att styra robotarnas beteende. Behandlingen av datan omfattar dels att lokaliserade objektens pixelkoordinater för exempelvis robotar och hinder. Dels hur pixelkoordinaterna omvandlas till världskoordinater i simuleringsmiljön. Därefter krävs ett beslutsfattande kring hur styrsignalen ska genereras baserat på den uppskattade omgivningen.

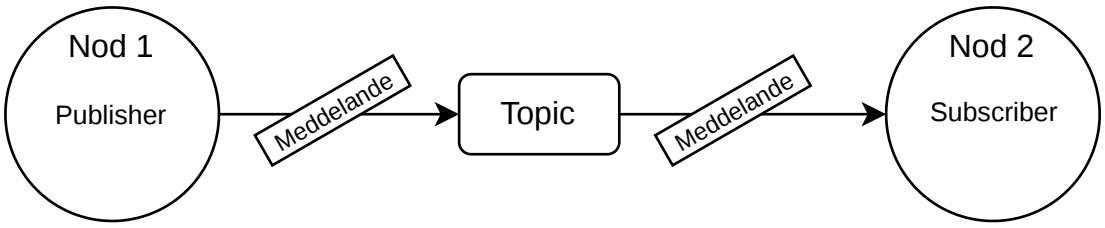
Detta projekt har valt att använda ett ramverk för att koppla detta system. Det ramverk som har valts är Robot Operating System 2 (ROS 2). Detta val har gjorts utifrån ansatsen i kapitel 2.4.2.



Figur 5.1: Systemöversikt av det kamerabaserade ROS 2-systemet, där bilder från flera övervakningskameror bearbetas genom segmentering, objektlokalisering och spårning, samt beslutsfattande för att styra en mobil robot och förhindra kollision med människor och andra hinder.

5.1 Introduktion till Robot Operating System 2

För att kunna integrera UE5 med segmenteringen, samt styra AMR:erna används Robot Operating System 2 (ROS 2). ROS 2 är ett bibliotek som underlättar byggandet av robotsystem, alltså ett helhetssystem som kopplar ihop styr- och ställdon [1]. ROS 2 bygger på en uppsättning av noder, där noderna kommunicerar med varandra genom topics, vilket är kommunikationskanaler. Noderna publicerar eller prenumererar (skicka eller ta emot) data från andra noder via dessa topics, se Figur 5.2.



Figur 5.2: ROS 2:s kommunikationssystem bygger på noder som kommunicerar genom topics. En nod kan publicera meddelanden till ett topic medan andra noder kan prenumerera på samma topic för att ta emot data.

Meddelanden kan vara av olika typer. De vanligaste typerna som kommuniceras är *String* och *Bool*. De meddelandetyper som är mest intressanta för detta arbete är *Image*, som möjliggör att bilder skickas över ROS 2. Dessutom är $Vector3$ av intresse, eftersom det är ett standardiserat sätt att skicka 3D-koordinater och likartad information. $Vector3$ meddelanden består av 3 komponenter, en X, Y och Z, bestående av flyttal.

Allt detta gör att ROS 2 fungerar som ett *middleware*, alltså mjukvara som möjliggör kommunikation mellan olika system. I detta fall mellan UE5, maskininlärningsalgoritmen (ML) och styrningsalgoritmer. Detta möjliggör att kamerabilder kan skickas från UE5 till ML-algoritmen och att styrsignaler kan skickas tillbaka till robotarna i UE5.

5.2 Integration mellan UE5 och ROS 2

För att möjliggöra integrationen mellan Unreal Engine 5 och ROS 2 krävs att data från simuleringen kan publiceras som ROS-kompatibla meddelanden, alltså meddelanden enligt samma protokoll. UE5 har inget inbyggt stöd för direktkommunikation med ROS 2, vilket innebär att det behövs något ett mellanliggande system. Därför används en brygga som möjliggör överföring av data från UE5 till ROS 2.

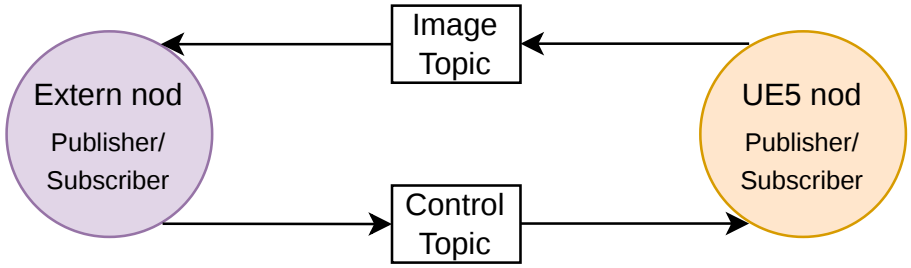
5.2.1 Brygga för dataöverföring

Den brygga som har valts att användas är TempoROS, en del av tilläggssamlingen Tempo, närmare beskrivet i avsnitt 2.4.3. TempoROS är ett tillägg i UE5 som möj-

[1] `URL:https://www.ros.org/`

liggör ROS 2-kommunikation genom rclcpp i UE5. Genom *ROSClientLibraryC++* (rclcpp) har UE5 tillgång till ROS 2:s bibliotek för C++, vilket möjliggör skapandet av ROS 2-noder i UE5, eftersom UE5 är kodat i C++ [2].

Vidare går det att publicera data från simuleringen till ROS-topics, eftersom UE5 blir en nod i ROS-systemet, enligt Figur 5.3. Detta möjliggör att simulerad data kan behandlas på samma sätt som verklig sensordata, eftersom ROS 2 bygger på ett standardiserat meddelandebaserat system där noder enbart interagerar med data och inte tar hänsyn till dess ursprung [3].



Figur 5.3: UE5-noden kommunicerar med den externa noden via ROS 2-topics, där TempoROS möjliggör att UE5 agerar som en vanlig ROS 2-nod i systemet. Exempelvis i denna figur kan bilder skickas via ett image topic och styrsignaler via ett control topic, mellan UE5 och externa noder.

5.2.2 Operativsystem

Vid integrationen mellan UE5 och ROS 2 uppstod ett problem kopplat till operativsystem, eftersom de olika delarna hade bäst stöd på olika plattformar. Initialt planerades allt att exekveras i Linuxmiljön, men flera delar av projektet saknade fullständigt stöd för Linux.

ROS 2 hade bäst stöd och stabilitet i Linuxmiljön, medan UE5 och de tillägg som användes i projektet krävde operativsystemet Windows. Detta innebar att systemet behövde utformas på ett sätt där simuleringen och kommunikationsramverket kunde köras på separata operativsystem, men ändå kunde kommunicera med varandra, [15].

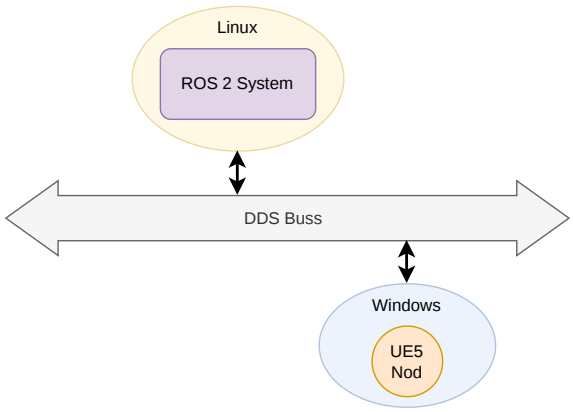
Det fanns två lösningar som projektet ansåg rimliga inom tidsramen. Den första lösningen var att använda Windows Subsystem for Linux (WSL) för att simulera Linux-operativsystemet på Windows. Tanken med denna lösning var att ROS 2 skulle exekveras i WSL-miljön samtidigt som UE5 kördes direkt i Windows, vilket skulle innebära att hela systemet skulle kunna köras på samma dator. Den andra lösningen var att använda två separata datorer, en med Windows och en med Linux, där respektive del av systemet exekveras på det operativsystemet där stödet och prestandan är bäst.

[2] `https://github.com/tempo-sim/Tempo`
[3] `https://github.com/tempo-sim/Tempo`

Ett problem som påträffades var att WSL tillsammans med ROS-systemet skapade hög latens, i praktiken blev det svårt att uppnå realtidskommunikation. Detta problem beskrivs även i WSL:s dokumentation och det är inte heller utformat för denna typ av ändamål [4].

Lösningen blev att använda separata datorer för systemet, eftersom detta gav lägre latens än att använda WSL vid tester. I den slutliga systemarkitekturen användes Windows för simuleringsmiljön i UE5, medan Linux användes för segmentering och exekvering av ROS 2.

Kommunikationen mellan systemen möjliggjordes genom en nätverksbaserad lösning via Ethernet där CycloneDDS användes. CycloneDDS är en implementation av DDS-standarden (Data Distribution Service), som används i ROS 2 för kommunikationen mellan noder [5]. När flera noder körs på samma dator fungerar DDS-kommunikationen internt mellan processerna, medan DDS vid distribuerade system, som när noder körs på olika datorer, även kan överföra data över nätverk mellan olika datorer. Detta sker genom att publicerad data delas upp i nätverkspaket som skickas över Ethernet, via en gemensam DDS-buss, se Figur 5.4 [6].



Figur 5.4: ROS 2-systemet på Linux och UE5-noden på Windows kommunicerar över en gemensam DDS-buss via Ethernet.

Detta möjliggjorde att bilddata från simuleringen kunde överföras till ROS 2 för vidare behandling, samtidigt som genererade styrsignaler kunde skickas tillbaka till simuleringen, på samma kommunikationsväg. Resultatet blev en lösning där delsystemen kunde köras i respektive optimal miljö.

[4] https://learn.microsoft.com/en-us/windows/wsl/compare-versions

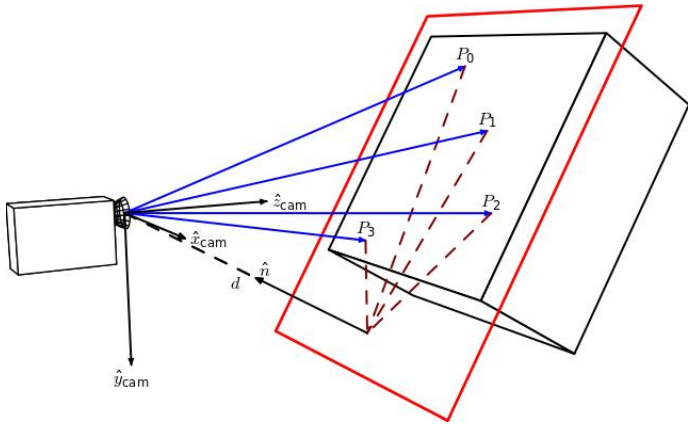
[5] https://docs.ros.org/en/foxy/Installation/DDS-Implementations/Working-with-Eclipse-CycloneDDS.html

[6] https://docs.ros.org/en/iron/Concepts/Intermediate/About-Different-Middleware-Vendors.html

5.3 Omvandling från pixel- till världskoordinater

Den information som extraheras från lokaliseringen av objekt består av koordinater uttryckta i pixlar från en kamera. Pixelkoordinater har ingen direkt koppling till positioner i den simulerade miljön och kan därför inte användas direkt för positionering av AMR:er. Systemet behöver därmed förhålla sig till två olika koordinatsystem. Ett koordinatsystem uttryckt i pixlar från kamerabilden och ett koordinatsystem uttryckt i världskoordinater från den simulerade miljön. Problemet blir därför att omvandla koordinater mellan dessa koordinatsystem.

Ett liknande problem inom verkliga AMR-system behandlas av Brorsson et al. [42]. Kameror observerar världen från ett perspektiv, vilket innebär att positioner i kamerabilden inte direkt motsvarar verkliga positioner i världen. Detta illustreras i figur 5.5, där punkter på ett plan (P_0, P_1, P_2, P_3) observeras från kamerans perspektiv och projiceras till bildplanet. Eftersom planet observeras i vinkel blir positionerna förvrängda i bilden jämfört med verkligheten.



Figur 5.5: Illustration av hur punkter på ett plan projiceras från verkligheten till kamerans bildplan. Bild baserad på Appoose och Per Rosengren via Wikimedia Commons [43].

Brorsson et al. beskriver att detta problem kan lösas genom homografi, där koordinater i bildplanet transformeras till ett gemensamt plan i världen. På så sätt blir det möjligt att omvandla pixelkoordinater till världskoordinater och därmed använda kamerabaserad perception för positionering.

Detta arbete har valt att använda samma princip som Brorsson et al. presenterar, men istället för att transformera koordinaterna från bildplanet till koordinater i den verkliga världen transformeras de till koordinater i den simulerade miljön [42]. Valet av metod motiveras av att problemen är nära relaterade samt att homografi är en etablerad och beprövad metod för projektiv transformation mellan plan.

Transformationen kan beskrivas med en 3×3-matris H, som relaterar en punkt (u, v) till en punkt (x, y), enligt ekvation 5.1. I detta fall kan (u,v) vara pixelkoordinater och (x,y) världskoordinater [44].

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = H \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{5.1}$$

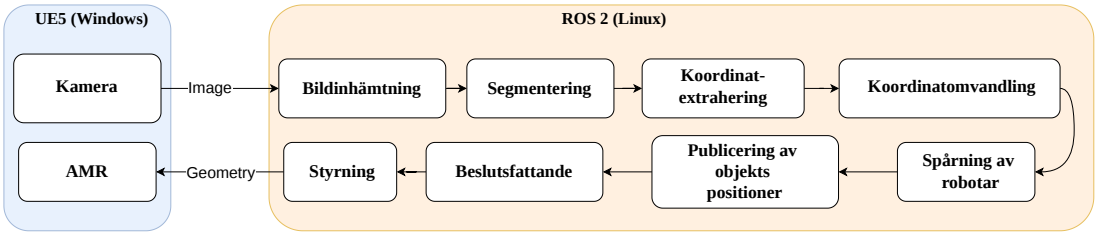
De slutliga koordinaterna erhålls genom normalisering enligt (x/w, y/w), vilket innebär att transformationen sker i homogena koordinater [44].

För att bestämma transformationsmatrisen H krävs ett antal referenspunkter med kända koordinater i både bild- och världsplanet [44]. När H har bestämts kan den användas för att omvandla nya punkter, exempelvis centroider av segmenteringsobjekt, som i detta projekt, från bildkoordinater till positioner i den simulerade miljön.

Metoden bygger på antagandet att de observerade objekten befinner sig på ett plan, i detta fall golvet, vilket möjliggör en tvådimensionell avbildning mellan bild- och världskoordinater [44].

5.4 Från lokalisering till positionering

De presenterade komponenterna kan nu kopplas samman till ett komplett system för perception och positionering av AMR:erna



Figur 5.6: Översikt av systemets pipeline från bildinhämtning och segmentering till positionering av AMR:er genom ROS 2.

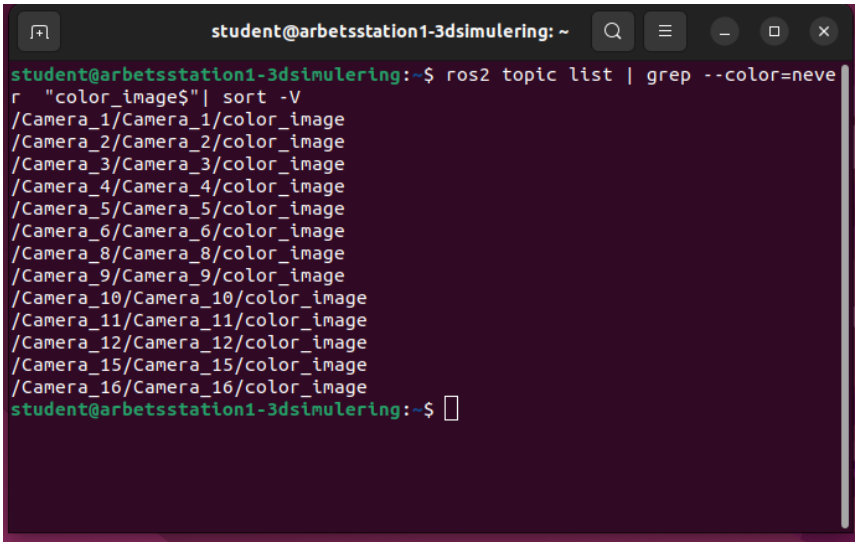
Systemet bygger på en pipeline där bilddata från simuleringsmiljön behandlas för att möjliggöra styrning av AMR:erna. Kamerabilderna från UE5 skickas till ROS 2, där segmenteringen och koordinatextraheringen används för att identifiera objekt i scenen. Därefter transformeras koordinaterna från bildplanet till världskoordinater, vilket möjliggör spårning av robotar och framtagning av beslutsunderlag för styrning, se figur 5.6. I följande avsnitt beskrivs de olika stegen i pipelinen mer ingående.

5.4.1 Insamling av data

För att möjliggöra vidare bildbearbetning behöver bilddatan överföras från simuleringsmiljön i UE5. Efter att de simulerade kamerorna har renderat bilder, publiceras dessa på ROS 2-topicen /camera_n/camera_n/color_image med hjälp av TempoROS, som beskrivs i 5.2.1, enligt Figur 5.7. Detta möjliggör överföring av bilddata från simuleringsmiljön på Windows till det externa ROS 2-systemet på Linux för

vidare behandling.

Bilderna representeras i ett standardiserat meddelandeformat av typen *Image* och innehåller rå pixelinformation, vilket innebär att bilderna är okomprimerade och representeras som numeriska värden för varje pixel, organiserade i RGB-format [7]. Slutligen lagras bilderna på en buffert om 10 och väntar på att bli inhämtade för behandling av segmenteringsnoden.



Figur 5.7: Topicsen för kamerorna i simuleringsmiljön som visas i Linux-terminalen. Där första */Camera_n* är publiceringsnodens namn, andra */Camera_n/* är kamerans namn, *color_image* beskriver att RGB-bilder skickas, varav n är kamerans id.

5.4.2 Behandling av data

Efter att bilddatan har samlats in behandlas den i flera steg för att få ut relevant information för att detektera hinder. Efter att hinder har detekterats fattas ett beslut om huruvida AMR:er bör stoppas eller om de kan fortsätta åka.

Bildinhämtning

Bilddata tas kontinuerligt emot från kamerorna i simuleringsmiljön på Windows och överförs till Linux-systemet för vidare bearbetning och behöver sedan omvandlas till rätt format för att kunna behandlas vidare. Rätt format i detta fall syftar till ett OpenCV-kompatibelt format. OpenCV används eftersom det ger en praktisk representation av bilden som en NumPy-array, vilket gör det möjligt att arbeta direkt med pixelvärdena i den fortsatta bearbetningen. Att formatera bilden enligt som Numpy-array är även ett krav för segmenteringsmodellen [8].

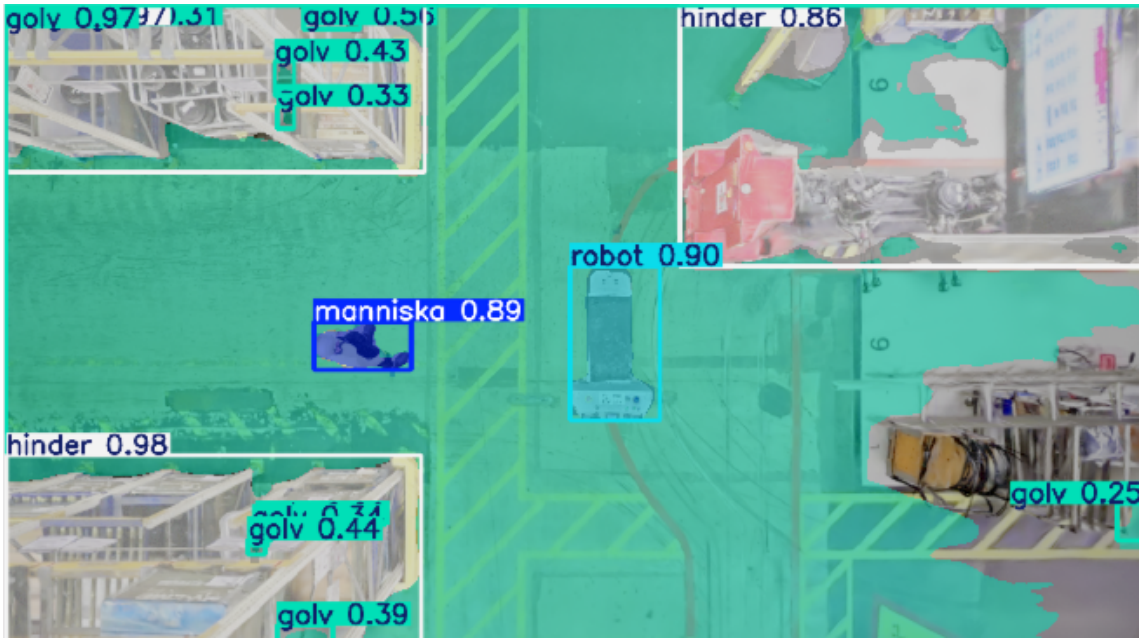
I samband med denna omvandling ändras även kanalordningen, alltså ordningen

[7] `https://docs.ros.org/en/jazzy/p/sensor_msgs/msg/Image.html`
[8] `https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html`

av färgerna, från RGB till (Blue Green Red), eftersom detta är något OpenCv använder internt. Efter att denna omvandling har genomförts kan bilderna segmenteras [9].

Segmentering

När bilderna nu representeras som en NumPy-array appliceras den tränade segmenteringsmodellen på varje bild. Modellen klassificerar varje pixel och genererar en segmenteringsmask där objekt såsom robotar, hinder, människor och väggar identifieras, se Figur 5.8. För mer information se kapitel 4. Detta möjliggör att relevanta delar av miljön kan användas för vidare analys.



Figur 5.8: Segmentering av bild från simulerad kamera. Mörkblå ruta visar en segmenterad människa. Ljusblå ruta visar en segmenterad robot. Vita och turkosa rutor visar segmenterade hinder respektive golv.

Framtagning av objektkoordinater

Utifrån segmenteringsmasken beräknas objektens position genom att bestämma (x_c, y_c), alltså mitten, för varje segmenterat område, enligt ekvationen 5.2. Denna centroid representerar objektets position i bildplanet och uttrycks i pixelkoordinater.

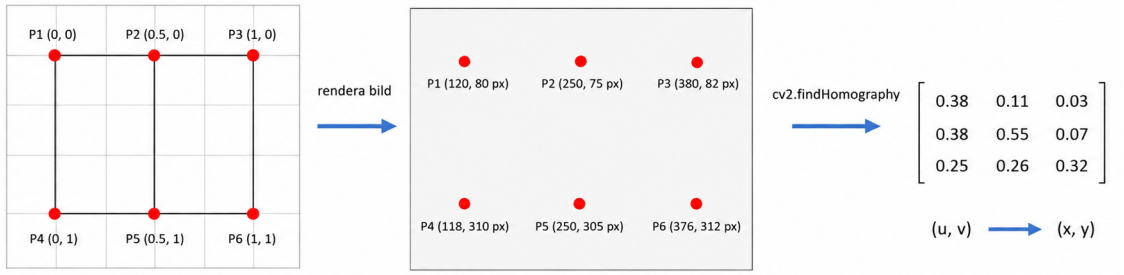
$$x_c = \frac{x_{min} + x_{max}}{2}, \; y_c = \frac{y_{min} + y_{max}}{2} \tag{5.2}$$

[9] `https://docs.opencv.org/4.x/db/deb/tutorial_display_image.html`

Figur 5.9: Framtagning av objektets centroid enligt ekvation 5.2. Variablerna x_{min} och x_{max} representerar objektets övre respektive nedre gräns i bilden, medan y_{min} och y_{max} representerar den vänstra respektive högra gränsen. Centroiden (x_c, y_c) beräknas som mitten av begränsningsrutan i pixelkoordinater.

Omvandling från pixel- till världskoordinater

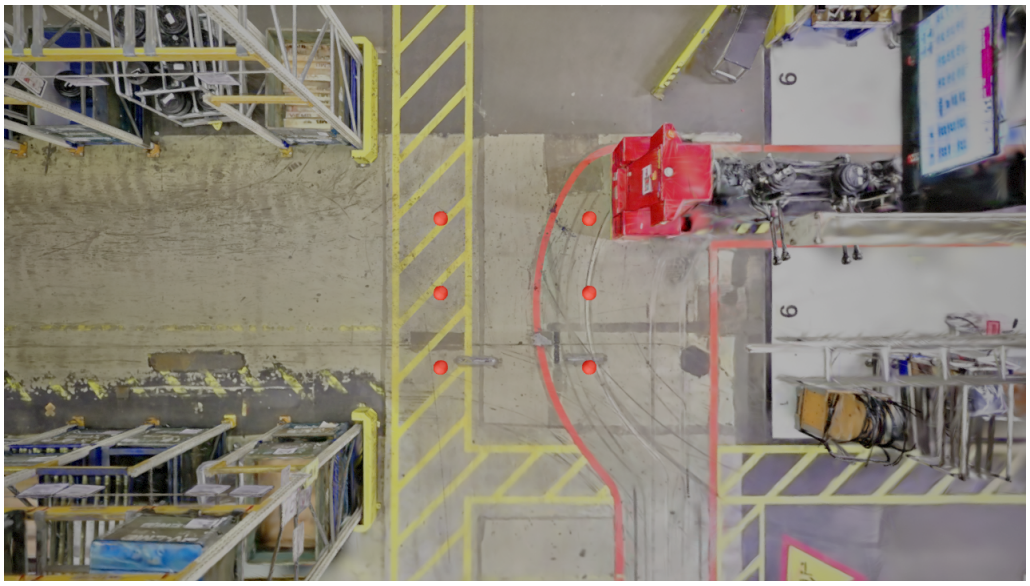
För att kunna använda positionsinformationen i styrningen omvandlas pixelkoordinaterna till världskoordinater med hjälp av en transformation, enligt 5.3. Transformationen erhålls genom kalibrering, där ett antal punkter med kända koordinater i både bild- och världsplanet används för att beräkna en transformationsmatris. I detta projekt används den fördefinierade funktionen från OpenCV för att utföra denna beräkning, enligt Figur 5.10.



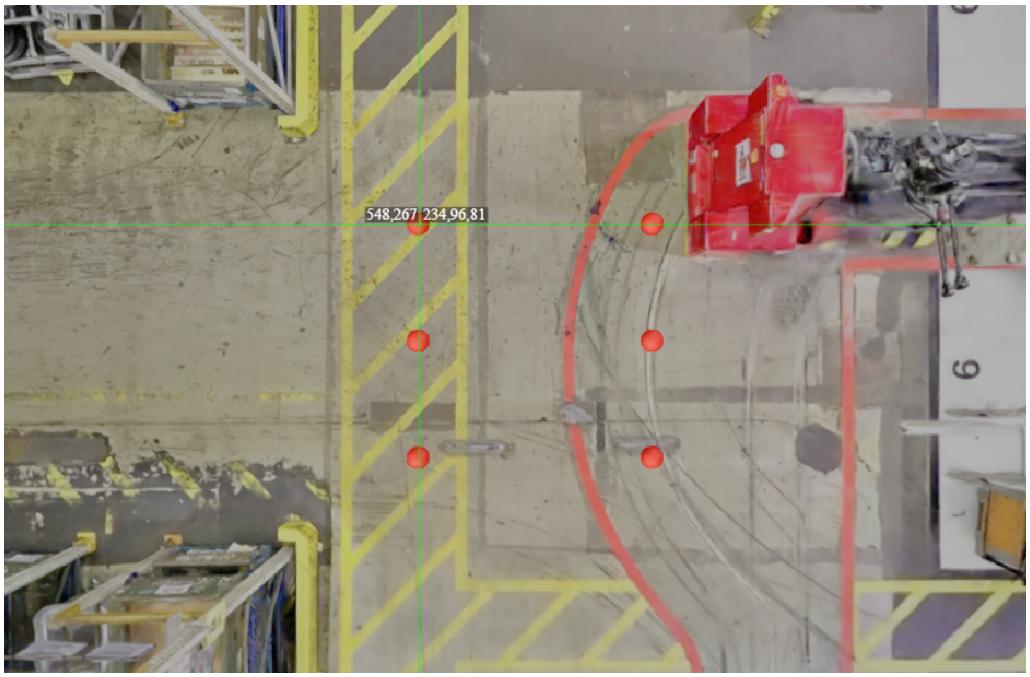
Figur 5.10: Exempel på hur homografi används för att beräkna en transformationsmatris mellan bildplanet och världsplanet. Genom att använda punkter med kända koordinater i båda koordinatsystemen kan OpenCV-funktionen cv2.findHomography beräkna en matris som möjliggör transformation från pixelkoordinater (u, v) till världskoordinater (x, y).

Denna princip användes genom att 6 referenspunkter placerades i simuleringsmiljön nära golvet, där deras koordinater noteras, enligt Figur 5.11. Sedan togs en bild med kameran och referenspunkternas pixelkoordinater antecknades, enligt Figur 5.12. Efter att detta gjordes kunde homografimatrisen skapas med hjälp av OpenCV:s beräkningsfunktion $findHomography$.

När homografimatrisen har definierats för kameran kan den multipliceras med nya punkter från bildplanet för att hitta deras motsvarande positioner i den simulerade miljön, enligt 5.3.



Figur 5.11: Referenspunkter använda vid kalibrering av homografin mellan bildplanet och världskoordinatsystemet. Punkterna markerades manuellt i kamerabilden och kopplades till kända världskoordinater i den simulerade miljön. De använda världskoordinaterna var $(-23, 50)$, $(-23, 250)$, $(-123, 40)$, $(-123, 250)$, $(-223, 50)$ och $(-223, 250)$. Dessa punktpar användes för att beräkna homografimatrisen för kameran.



Figur 5.12: Kalibreringspunkter markerade i kamerabilden. Punkterna användes för att skapa motsvarande punktpar mellan pixelkoordinater i bildplanet och kända världskoordinater i den simulerade miljön. De extraherade pixelkoordinaterna var $(548, 267)$, $(733, 267)$, $(546, 361)$, $(733, 361)$, $(546, 454)$ och $(733, 453)$.

Spårning av robotar

När positionerna för de lokaliserade objekten har tagits fram har projektet valt att identifiera och spåra robotarna över tid. Detta eftersom systemet då kan följa robotarnas rörelse i miljön och fatta beslut om framtiden. Det är också nödvändigt för styrsystemet att veta robotens position i realtid för att skicka rätt styrsignal.

Spårningen av robotarna bygger på närhetsmatchning. För varje ny detekterad av robot beräknas avståndet till alla tidigare spårade robotar och det närmaste spåret väljs. Om avståndet är mindre än ett fördefinierat tröskelvärde antas det vara samma robot och den tidigare positionen uppdateras med den nya. Om ingen tidigare robot ligger tillräckligt nära, skapas i stället ett nytt robot-ID. På så sätt kan systemet följa samma robot över flera uppdateringar även när nya bildrutor kommer in.

Detta är en relativt enkel spårningsmetod, men på grund av projektets tidomfång sågs detta som det mest realiserbara alternativet.

Vektorpublicering

När världskoordinater för alla objekt som segmenteringsmodellen har identifierat har tagits fram och klassificerats, kan dessa publiceras för att användas i ett beslutsfattandet.

Koordinaterna publiceras med datatypen *Vector*3, som består av X-, Y- och Z-komponenter, på topicen `/camera_n/vector_3`. I denna representation används X- och Y-komponenterna för att ange objektets position i planet, medan Z-komponenten används för att annotera objektets klass, exempelvis robot eller människa. Där Z-komponenten för robotar är dess id, medan Z-komponenten för hinder är -1. Detta för att kunna skilja på vad som är hinder och robotar. Detta gör att både positionsinformation och klassificering kan överföras i samma meddelande, vilket gör att separata topics inte behövs för varje objekt.

5.4.3 Styrning av mobil robot

När alla koordinater för samtliga hinder och robotar har samlats in är det möjligt att fatta ett beslut kring de mobila robotarnas styrsignal. Om ett hinder är för nära AMR:en skall den stanna och om inte ska den köra.

Beslutsfattande kring reglering av AMR

Baserat på objektets position analyseras robotens omgivning. En säkerhetszon definieras kring roboten, ett avstånd från roboten där närvaro av hinder eller andra objekt inom zonen ska leda till att roboten stoppas.

Om ett hinder på koordinaterna (x_1, y_1) är för nära en robot på koordinaterna (x_2, y_2) bestäms med den euklidiska avståndsformeln, enligt ekvation 5.3.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{5.3}$$

Om d hamnar inom den fördefinierade säkerhetszonen behöver roboten stanna och ett stoppbeslut skapas.

Stoppmeddelande-publicering

När ett objekt har identifierats inom den definierade säkerhetszonen genereras ett stoppbeslut. Detta skapar ett stoppmeddelande som publiceras på det dedikerade ROS 2-topicet, `/robot_n/stopp_command`. Även detta meddelande använder datatypen $Vector3$, som i 5.4.2, fast i detta fall med X-komponenten som definierar om roboten får köra eller inte.

Styrning

För att sedan styra AMR:erna användes tillägget TempoMovement, som erbjuds av den tidigare nämnda tilläggssamlingen Tempo. Med hjälp av TempoMovement kan `Pawns` styras, en karaktärstyp i UE5, genom funktionen `pawn_move_to_location`. Denna funktion möjliggör styrning av Pawns genom teleportering till godtyckliga världskoordinater inom den spelbara simuleringsytan, se avsnitt 3.5.3. För att åstadkomma detta använder sig funktionen av Tempos egna topic `Geometry` och UE5:s inbyggda navigeringssystem.

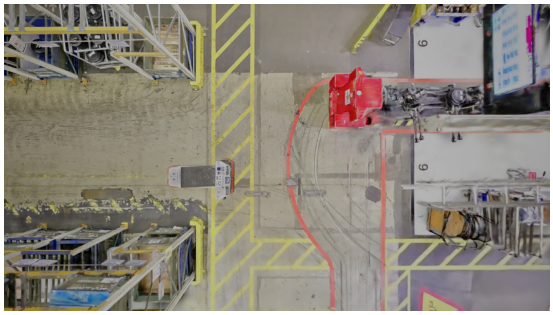
Det första steget var därför att manipulera de AMR:er, som beskrevs i avsnitt 3.5.2, så att de inte längre var av klassen karaktär utan istället `Pawn`. Därefter krävs det att en ny rörelsekomponent adderas. I detta arbete valdes rörelse komponenten `Floating Pawn Movement`.

Därefter implementerades en styrnod som bygger på den tidigare nämnda styrfunktionen från TempoMovement. Denna nod prenumererar sedan på ROS 2-topicet `/robot_n/stopp_command` för att kontinuerligt ta emot information om huruvida AMR:en ska stanna. Eftersom banplanering ligger utanför detta arbetes omfattning förutsätter styrnoden att den förses med en lista bestående av glest utspridda måldestinationer. Då styrfunktionen, som tidigare nämnts, förflyttar AMR:en mellan diskreta positioner används sedan interpolering, en matematisk metod för uppskattning av värden mellan känd data, för att uppskatta en mängd tätt närliggande destinationer mellan de utspridda måldestinationerna. På så sätt kan en approximativ väg mellan samtliga måldestinationer uppskattas

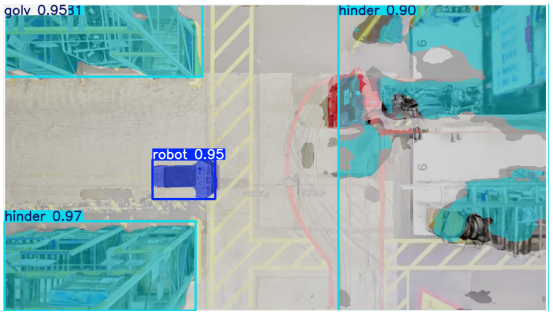
5.5 Utvärdering

Målet med detta delproblem var att implementera en metod för att omvandla bilddata från kameraperception till styrsignaler för AMR:er och i detta avsnitt utvärderas resultatet.

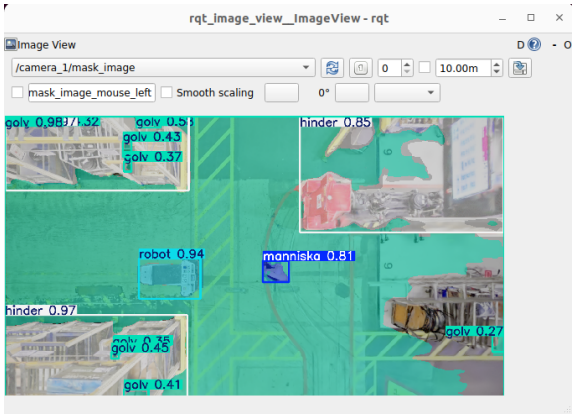
Den realiserade pipelinen visar att det är möjligt att koppla samman perception och styrning i ett modulärt system som ROS 2. Lokalisering av objekt fungerar och pipelinen från bilddata till generering av objektkoordinater fungerar i huvudsak som avsett. Bilddatan kunde överföras från UE5 till ROS 2 där RGB-bilder och segmenterade bilder behandlades. Objekt såsom robotar, människor och hinder kunde identifieras genom segmentering. Figur 5.13a, 5.13b och 5.13c visar exempel på hur bilddata överförs och behandlas i systemet.



(a) RGB-bild från kameran i UE5 som överförs till ROS 2-systemet för vidare behandling.



(b) Segmenterad bild där objekt såsom robotar, människor och hinder identifieras genom perception.

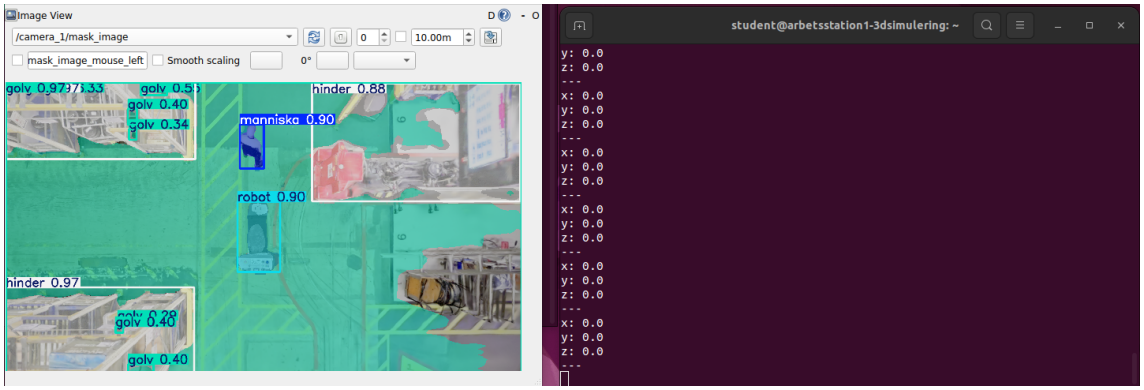


(c) Visualisering av bilddata i ROS 2 genom `rqt_image_view`.

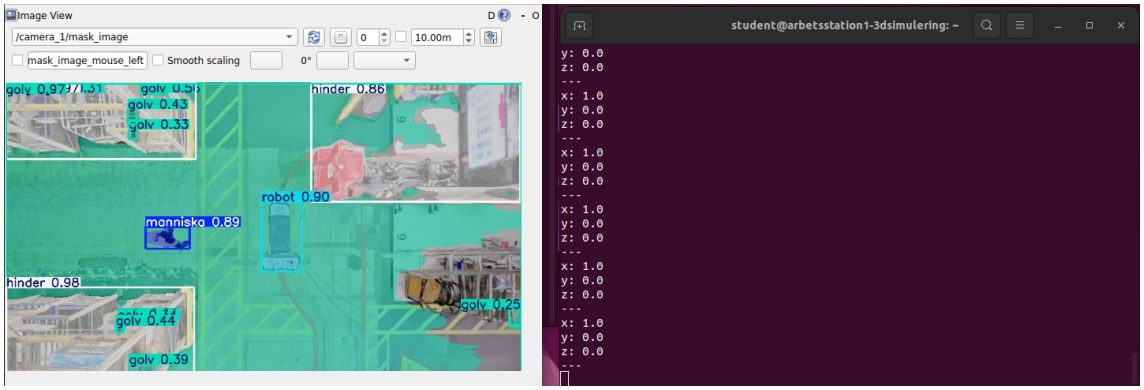
Figur 5.13: Exempel på hur bilddata överförs och behandlas i systemet, från RGB-bild i UE5 till segmentering och visualisering i ROS 2.

Vidare kunde systemet generera enkla styrbeslut baserat på den detekterade omgivningen. När en människa befann sig inom ett definierat säkerhetsområde publicerades ett stoppmeddelande. Detta visar att det är möjligt att koppla perception och styrning i ett ROS 2-baserat system. Figur 5.14 visar hur ett stoppmeddelande

genereras när en människa är nära roboten, medan 5.15 visar att roboten inte får ett stoppmeddelande om människan inte är lika nära.



Figur 5.14: Stoppsignal publiceras när människan befinner sig inom det definierade säkerhetsområdet kring roboten. X-komponenterna i meddelandet är 0, vilket representerar att roboten ska stanna.

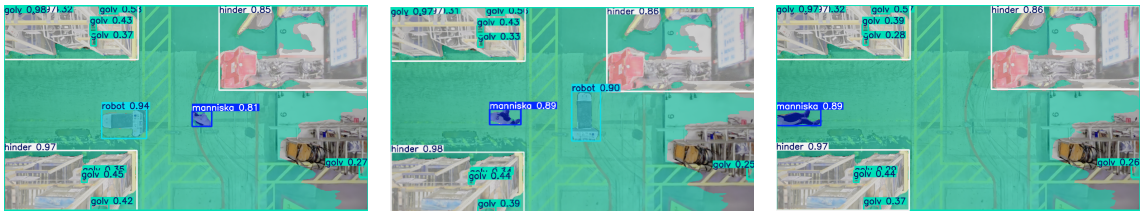


Figur 5.15: Ingen stoppsignal genereras när människan befinner sig utanför det definierade säkerhetsområdet kring roboten. X-komponenterna i meddelandet är 1, vilket representerar att roboten ska köra.

Resultatet visar även att homografin i teorin är en användbar metod för att omvandla pixelkoordinater till världskoordinater, vilket möjliggör en direkt koppling mellan bildplanet och den simulerade miljön. Tester visar dock att precisionen varierar beroende på objektets position i bilden, se tabell 5.1.

Tabell 5.1: Avvikelse mellan beräknad och verklig position vid olika placeringar i kamerabilden. Positionerna motsvarar placeringarna som visas i Figur 5.16a, 5.16b och 5.16c.

Position av människa	Beräknad position (cm)	Verklig position (cm)	Avvikelse (cm)
I centrum	(-177.4, -212.0)	(-176.3, -205.3)	6.8
Mellan långt från centrum	(-178.0, -85.6)	(-177.0, -75.6)	10.0
Långt från centrum	(-174.4, 59.4)	(-172.3, 72.4)	13.2



(a) Människa (mörkblå) placerad nära centrum av kamerabilden.

(b) Människa (mörkblå) placerad längre från centrum.

(c) Människa (mörkblå) placerad längst från centrum av kamerabilden.

Figur 5.16: Positioner som användes vid utvärdering av homografin. Bilderna visar hur objektets placering i kamerabilden påverkar precisionen vid omvandling från pixelkoordinater till världskoordinater.

Resultatet indikerar att precisionen försämras längre från det område där homografin kalibrerades, enligt Figur 5.11. Detta beror sannolikt på perspektivdistorsion från kamerans synfält på 90 grader, där små fel i pixelkoordinater ger större fel i de beräknade världskoordinaterna längre ut mot bildens kanter. Detta gör homografibaserad positionering känslig för kalibreringsfel och brus. En möjlig förbättring hade varit att använda en mer avancerad kamerakalibrering, exempelvis chessboard-baserad kalibrering, för att kompensera för linsdeformation och perspektivfel, [10].

Fortsättningsvis fungerar flera delar av systemet i sig och detta visar att perception kan användas för att generera enkla styrbeslut i ett ROS 2 baserat system. Lokalisering av objekt, överföring av bilddata och generering av objektkoordinater fungerar i huvudsak som avsett. Även koordinatomvandlingen fungerar relativt bra, särskilt nära det området där kameran kalibrerades, även om precisionen försämras längre från centrum av kamerabilden.

Samtidigt finns det flera problem i systemet. Den använda metoden för spårning är relativt enkel, vilket försvårar hanteringen av flera robotar samtidigt. Mer avancerade metoder hade sannolikt förbättrat systemets stabilitet.

Den fullständiga autonoma styrningen realiserades aldrig under projektet. Systemet kunde generera stoppmeddelanden, men dessa integrerade sig inte fullt ut med roboternas rörestyrning i UE5, utan testades separat. Detta innebär att hela pipelinen för autonom reglering inte kunde verifieras fullt ut inom projektets tidsram.

Sammanfattningsvis visar resultaten att perception kan användas för att lokalisera objekt och generera enklare styrbeslut i ett ROS 2-baserat system. Samtidigt visar resultaten att precisionen i koordinatomvandlingen påverkas av kamerakalibreringen och objektets position i bilden. För att skapa en komplett autonom lösning krävs mer robust styrning, förbättrad kalibrering och mer avancerade metoder för spårning och pathfinding.

[10] `https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html`

5.6 Sammanfattning

Sammanfattningsvis visar arbetet kring detta delproblem att det är möjligt att koppla samman perception och styrning i en simulerad miljö genom ett distribuerat ramverk som ROS 2. Flera delsystem fungerar enligt avsikt, men integration och förenklad styrlogik begränsar systemets totala prestanda. Arbetet utgör därmed en grund för vidare utveckling mot mer avancerad autonom styrning.

6
Utvärdering

I detta kapitel presenteras och utvärderas först helhetslösningen, den digitala verkligheten, som sammanfogar samtliga arbetsprocesser som beskrivits i kapitel 3, 4 och 5. Därefter evalueras samtliga arbetsprocesser gentemot tillhörande frågeställningar som ställts i avsnitt 1.4.

6.1 Digital verklighet

Den digitala verkligheten består av följande komponenter: En fotorealistisk simuleringsmiljö, lokalisering med segmenteringsmodeller och styrning med perception. I detta arbete samverkar dessa komponenter för att möjliggöra simulering av den fysiska fabriksmiljön på AB volvo i Tuve. Vid framtagningen av den digitala verkligheten ligger fokuset främst på att möjliggöra för en verklighetstrogen simulering av den fysiska fabriksmiljön

För att åstadkomma det här innehåller den fotorealistiska simuleringsmiljön både statiska objekt och dynamiska karaktärer, likt de som kan påträffas i den fysiska fabriksmiljön. Därutöver innehåller även simuleringsmiljön takmonterade digitala kameror. Dessa kameror tillåter sedan för digital bildtagning av både de statiska objekt och dynamiska karaktärer som befinner sig i simuleringsmiljön. I sin tur möjliggör den digitala bildtagningen för insamling av bilddata som kan användas vid såväl fristående träning av segmenteringsmodeller och övervakning av fabriksmiljön. Här genomförs övervakningen av miljön med hjälp av en segmenteringsmodell från den fristående träningen. Baserat på den bildata som skickas ifrån de takmonterade digitala kamerorna kan sedan en prediktion, i form av en segmenteringskarta, från segmenteringsmodellen utvinnas. Denna prediktion används sedan vid beslutsfattande kring styrning av de dynamiska karaktärerna i simuleringsmiljön.

Under arbetes gång identifierades en del för- och nackdelar kopplat till samspelet mellan alla dessa beståndsdelar. Fördelen med att nyttja en såpass vedertagen spelmotor, UE5, är det breda utbudet av tillägg samt funktionaliteter som detta medför. I sin tur leder det här till en god förmåga att modellera en bred mängd olika scenarier som kan tänkas förekomma i den fysiska verkligheten. Dessvärre begränsas denna förmåga av bildtagningsprocessen för att skapa segmenteringskartor. Vid simulering av den uppbyggda simuleringsmiljön påträffades det att håret för de mänskliga karaktärerna, implementerade med hjälp av MetaHumans, renderades med viss fördröjning. Anledningen till denna begränsning tros bero på att den

funktionalitet som bildtagningsprocessen använder sig av för att spara bilddata, `USceneCaptureComponent2D`, exekveras före det att håret hinner uppdateras under varje bildsekvens. På så sätt reduceras även variationen hos den träningsdata som kan samlas in av de digitala kamerorna, då återspeglingen av mänskliga karaktärer i simuleringsmiljön begränsas enbart till modeller utan hår. Detta medför en risk för försämrad generalisering hos segmenteringsmodellen, vilket innebär sämre prediktioner av informationen i bilddata. En sämre prediktion påverkar i nästa led styrningen av de dynamiska karaktärerna negativt. Detta är problematiskt då det reducerar omfattningen av de scenarier som kan återskapas i den digitala verkligheten, vilket ökar sim-to-real gapet.

På ett liknande sätt uppstod även svårigheter vid simulering av en specifik typ av ovanlig händelse, nämligen eld. Även här härstammar dessa svårigheter från bildtagningsprocessen, då den inte stödjer funktionaliteten att segmentera eld. Avsaknaden av denna funktionalitet tros bero på att elden representeras av effekter i simuleringsmiljön, vilka inte har förmågan att representeras av ett Custom Stencil ID. Avsaknaden av denna funktionalitet hos den digitala bildtagningen anses även i detta fall vara problematiskt då det, likt tidigare, i förlängningen kommer reducera omfattningen av de scenarier som kan återskapas i den digitala verkligheten, vilket ökar sim-to-real gapet.

Samtidigt bör det noteras att den digitala bildtagningsprocessen vid ett tillfälle under arbetes gång renderade totalt 4500 bilder med bildupplösningen 640x360 på fem minuter, vilket resulterar i renderingshastighet på 15 bilder per sekund.

6.2 Fotorealistisk 3D-rekonstruktion och simulering

Vid skapandet av en realistisk simuleringsmiljö var målet att utvärdera hur 3DGS kan användas för att skapa miljön. Målet härstammar från arbetets frågeställning: Vad är möjligheterna och begränsningarna med att använda 3D Gaussian Splatting för att skapa en realistisk simuleringsmiljö?. För att utvärdera detta jämförs simuleringsmiljön med den verkliga motparten.

3DGS är ett användbart verktyg som gör det enkelare att skapa realistiska 3D miljöer av verkligheten. Formatet tillåter även hög bildfrekvens vilket tillåter miljön att bli interaktiv. Däremot finns utmaningar vid implementering i UE5 och användning för mer komplexa ändamål än att enbart visualisera miljön. Ett problem är att 3DGS saknar fysiska egenskaper som är avgörande för en fungerande simulering och kräver då att separata polygonytor adderas, vilket komplicerar arbetsflödet. Tekniken är ännu relativt ny, tydligast märks detta genom avsaknaden av inbyggda verktyg för 3DGS. Avsaknad av vedertagna arbetssätt orsakar frekventa kompatibilitets problem. För att kringå dessa krävs ofta komplicerade eller hämmande lösningar.

6.3 Segmentering och maskininlärning

Syftet här är att besvara forskningsfrågan: för- och nackdelarna med att använda syntetisk träningsdata för lokalisering av objekt i simuleringsmiljö? För att besvara detta utvärderas måtten IoU per klass, mIoU, mAP50 samt mAP50-95 av den tränade modellen YOLOv8-Seg Nano, som beräknades i avsnitt 4.2.4.

I arbetet har det visats att insamling av stora mängder syntetisk träningsdata är möjlig genom UE5. Annotering av data kunde automatiseras med hjälp av spelmotorn vilket underlättar processen gentemot hur datainsamling från verkligheten hade skett. Resultaten tyder även på att datan hade tillräcklig variation av scener eftersom många av objekten i verkliga bilder kunde kännas igen. Därtill visade modellen stabila värden på mAP50, mAP50-95 samt mIoU. Detta tyder på att syntetisk träningsdata kan användas till lokalisering av objekt i även verkliga bilder.

Däremot identifierades några svagheter med den syntetiska träningsdatan. En nackdel var att den tränade YOLOv8-Seg Nanos IoU-värde av klassen människa var endast 0,009 vilket pekar på att träningsdatan hade för lite bilder med människor. När modellen testades på syntetiska test-bilder kunde modellen identifiera människor. Vid testning på en verklig bild innehållandes människor, lyckades modellen däremot inte identifiera dessa. Därför bedöms det, att utöver att datasamlingen innehöll för få inslag av människor, att en annan begränsning var att modellen tränades med enbart människor av samma utseende. Detta är alltså inte en realistisk representation av alla människor som förekommer i industrimiljön i verkligheten, vilket kan i sin tur påverka robot-människa interaktionen i simuleringsmiljön och följaktligen i verkligheten. En begränsning med denna bedömning är dock att modellen testades på endast en verklig bild innehållandes människor, vilket inte är en utförlig representation av modellens förmåga. Men genom användning av simulerade människor istället för verkliga människor till träning av segmenteringsmodeller kan etiska aspekter underlättas.

En till fördel med syntetisk träningsdata för lokalisering av objekt är att ovanliga händelser kan simuleras. Bränder eller pallar som blivit felplacerade kunde simuleras, och filter med repor, smuts och ljusskiftningar kunde appliceras på bilder för att efterlikna skador på kameror. Resultaten av de segmenterade bilderna indikerade att modellerna var sämre på att känna igen objekten med dessa filter. Detta beror på att träningsdatan inte innehöll denna sorts bilder utan att dessa bilder användes endast vid testning.

Sammanfattningsvis bedöms användningen av syntetisk träningsdata för lokalisering av objekt i simuleringsmiljön vara bra men den befintliga syntetiska träningsdata har otillräcklig representation av människor eller ovanliga händelser.

6.4 Perception och styrning

Vid styrning av autonoma mobila robotar låg fokus på att utvärdera hur väl kamerabaserad perception kunde användas för att lokalisera objekt och generera styrbeslut i simuleringsmiljön. Utvärderingen baserades främst på systemets förmåga att identifiera människor, robotar och hinder genom segmentering, omvandling av objekts positioner från pixel- till världskoordinater, samt använda denna information för att styra robotar i realtid. Detta kopplas till arbetets frågeställning kring styrkor och svagheter med att använda perception för styrning av mobila robotar.

För att undersöka detta utvecklades ett ROS 2 baserat system där bilddata från kameror i UE5 segmenterades och användes för positionsbestämning genom homografi. Objektens positioner användes därefter för att generera styrsignaler till robotarna exempelvis en stoppsignal när en människa befann sig i ett fördefinierat område kring roboten.

Resultatet visade att perception kunde användas för att identifiera och lokalisera objekt i simuleringsmiljön, samt möjliggöra enklare styrning av AMR:erna. En styrka med metoden är att flera robotar kan övervakas samtidigt genom externa kameror utan behov av sensorer monterade på robotarna. Den modulära lösningen med ROS 2 möjliggör även att perception och styrning kunde integreras i ett gemensamt system.

Däremot identifieras flera begränsningar. Precision i positionsbestämmandet varierade beroende på var i bilden objektet befann sig och systemet var känsligt för fel i segmenteringen. Vidare kvarstod problem kopplade till robust spårning av robotar mellan flera kameror, vilket försvårar kontinuerlig övervakning i större miljöer. En annan begränsning var att styrningen endast omfattade enklare beslutsfattande, exempelvis stoppfunktioner, medan mer avancerad navigering och samordning mellan robotar låg utanför arbetes omfattning.

Arbetet visar trots detta potentialen att använda perception och digital tvilling för styrning av framtida AMR-system i industrimiljöer, där utveckling och utvärdering kan utföras i simulerade miljöer innan en verklig implementering.

7

Diskussion och slutsats

Projektet användningsområden och utvecklingsmöjligheter diskuteras samt hur det kan påverka samhället ur etniska och samhällsenliga perspektiv.

7.1 Framtida utveckling

Ett viktigt område för fortsatt arbete är att skapa större variation bland de digitala människornas utseende i form av kroppsform, längd, hår, kläder och hudfärg. Syftet med att skapa större variation är att bidra till en mer robust segmenteringmodell, bättre lämpad för att hantera verkliga bilder. För att åstadkomma detta behöver antingen träningsbilderna sparas efter att håret renderat klart, alternativt behöver håret visualiseras på något annat sätt än med UE5:s groom komponent.

I framtida arbete kan det vara intressant undersöka tillämpning av panoptisk segmentering, då man kombinerar instansindelning med den semantiska segmenteringen. Det som är fördelaktigt med semantisk segmentering som kan utnyttjas med panoptisk segmentering är att golvet skulle kunna behandlas som ett amorft område, alltså ett enda objekt. Golvet kan användas till styrning eftersom autonoma robotar skulle kunna använda det som ett stabilt navigerings område.

Sedan uppkomsten av 3DGS har flera närbesläktade format tillkommit. Ett format som är särskilt intressant för detta arbetet är formatet Meshsplatting[45]. Meshsplatting och Gaussian splatting bygger till stor del på samma principer. Det som skiljer teknikerna åt är att Meshsplatting nyttjar trianglar snarare än ellipsoider som 3DGS gör. Resultatet av Meshsplatting blir en fotorealistisk polygonyta som lämpar sig mycket bättre för program för spelutveckling. Flera av de problem som identifierades med 3DGS som visualiseringsformat i UE5 är icke-frågor utifall punktmolnet var en polygonyta likt vad Meshspaltting erbjuder. Den främsta fördelen är att miljön direkt kan skriva till CustomStencilBuffer. Det medför att de modeller som i detta arbete representeras av både en 3DGS-modell och en polygonyta kan åstadkommas med enbart en Meshsplatting modell. Det innebär också att det i teorin inte finns någon avvikelse mellan färgbild och och masken given av CustomStencil. Meshsplatting stödjer även renderings metoden ray tracing vilket innebär att simulering av skuggor och ljus underlättas. Meshsplatting har altså potential att både förenkla simuleringsmiljön samtidigt som Sim-To-Real gapet minskas.

I detta arbete implementerades endast grundläggande styrning av den mobila robo-

ten i form av stoppsignaler vid upptäckta hinder. En möjlig vidareutveckling är därför att implementera mer avancerade styralgoritmer för exempelvis path planning. Vidare skulle systemet kunna förbättras genom stabilare och mer exakt positionering av objekt. Eftersom styrningen är direkt beroende av perceptionens kvalitet påverkar fel i segmenteringen och positioneringen även robotens beteende. Mer avancerade metoder för spårning, såsom Kalmanfilter eller liknande tekniker och sensorfusion skulle därför kunna öka systemets robusthet.

7.2 Samhällsetiska aspekter

Projektet består av simuleringar gjorda med 3DGS av AB Volvos fabrik i Tuve. De samhälleliga och etiska frågorna som behandlas rör integritet och automatisering.

Vid inspelning av sektionerna i fabriken fanns det arbetare som rörde sig och kan ha oavsiktligt fångats i inspelningen. Arbetarna finns inte med i simuleringsmiljön då kameran ignorerar objekt i rörelse. Inspelningsfiler som lagrats i kamerans minne kan innehålla känslig information. För att undvika att materialet sprids bör inspelningarna på kameran raderas när projektet är avslutat.

Ett annat problem relaterat till miljöns användningssyfte är att kamerorna i taket spelar in kontinuerligt, för navigering och koordinering av AMR:er. Sådan information brukar traditionellt sett skickas till en central server för att kunna träna flera autonoma robotar på olika platser. Om filmmaterialet hanteras för felaktiga ändamål riskerar det att sprida känslig information. En möjlig lösning kan vara federerad maskininlärning. Då skickas endast modellens uppdaterade parametrar till den centrala servern och data sparas enbart lokalt [46] för att undvika att känslig information sprids.

En ytterligare aspekt som behöver övervägas är hur automatisering påverkar industrin och dess arbetare. Automatisering av industrin kan förbättra arbetsmiljön och utföra de tunga och farliga uppgifter som för närvarande utförs av mänsklig arbetskraft. Men detta kan medföra att den mänskliga arbetskraften konkurrerar mot effektiva robotar. Detta kan leda till minskade arbetsmöjligheter, vilket i sin tur kan resultera i ökad arbetslöshet. Detta riskerar i sin tur att orsaka missnöje i samhället [47]. Därtill kan även produktionen effektiviseras och öka, vilket kan resultera i att konkurrerande företag som består av mänsklig arbetskraft behöver sänka de anställdas löner. För att kunna bibehålla en låg kostnad mot den plötsligt högre produktionen. Detta kan orsaka klasskillnader i samhället. Även de arbetsuppgifter som finns kvar riskerar att bli för standardiserade och därför blir monotona och mindre stimulerande för de anställda, vilket kan leda till sänkt arbetsmotivation [48].

7.3 Slutsats

Arbetet visar att 3D Gaussian Splatting kan användas för att skapa en fotorealistisk digital tvilling av verkliga industrimiljöer. Genom implementering i UE5 kan en dynamisk simuleringsmiljö utvecklas där människor, hinder och AMR:er kan simuleras i realtid. Simuleringsmiljön möjliggjorde skapandet av automatiskt annoterad träningsdata.

Segmenteringsmodeller tränade på den syntetiska datan kan användas för att lokalisera och identifiera objekt med kamera perception. Resultaten visar att perception kan användas för att identifiera människor, robotar och hinder i simuleringsmiljön samt till viss del generalisera till verkliga industrimiljöer. Genom att kombinera segmentering, homografi och ROS 2-baserad kommunikation kan perception vidare användas för att generera styrbeslut för AMR:er, exempelvis stopp då människor eller hinder uppträder inom säkerhetsområdet.

Trots begränsningar visar arbetet potentialen i att använda digitala tvillingar för att utveckla, träna och utvärdera autonoma robotsystem i digitala miljöer istället för direkt i verkliga fabriker. Detta knyter an till arbetets grundproblem kring tidskrävande datainsamling och manuell annotering och visar hur digitala tvillingar kan bidra till en mer effektiv utveckling av framtidens AMR-system i moderna industrimiljöer.

Källförteckning

[1] E. Brorsson m. fl. ”Infrastructure-based Autonomous Mobile Robots for Internal Logistics – Challenges and Future Perspectives.” arXiv: 2512.15215 [cs.RO], hämtad 10 febr. 2026. URL: https://arxiv.org/abs/2512.15215.

[2] V. Group. ”AI transporters push the boundaries of modern manufacturing,” hämtad 10 febr. 2026. URL: https://www.volvogroup.com/en/news-and-media/news/2024/nov/ai-modern-manufacturing.html.

[3] H. Ahmed, L. Bodin, E. Jansson, L. Nilsson, M. Sjöstedt och D. H. Wallander. ”Realistisk träningsdata med 3D Gaussi-an Splatting Bygga digitala tvillingar för generering av ground-truth-data i Unreal Engine 5 Kandidatarbete inom Elektroteknik. ”URL: https://odr.chalmers.se/server/api/core/bitstreams/b5c8d13a-aad2-4570-b30e-d075f316d21f/content.

[4] M. W. Grieves, ”Digital twins: past, present, and future,” i *The digital twin*, Springer, 2023, s. 97–121.

[5] 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.

[6] J. V. Hurtado och A. Valada, ”Chapter 12 - Semantic scene segmentation for robotics,” i *Deep Learning for Robot Perception and Cognition*, A. Iosifidis och A. Tefas, utg., Academic Press, 2022, s. 279–311, ISBN: 978-0-323-85787-1. DOI: https://doi.org/10.1016/B978-0-32-385787-1.00017-8. URL: https://www.sciencedirect.com/science/article/pii/B9780323857871000178.

[7] A. M. Hafiz och G. M. Bhat, ”A survey on instance segmentation: state of the art,” *International journal of multimedia information retrieval*, årg. 9, nr 3, s. 171–189, 2020.

[8] A. Kirillov, K. He, R. Girshick, C. Rother och P. Dollár, ”Panoptic Segmentation,” i *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, s. 9396–9405. DOI: 10.1109/CVPR.2019.00963.

[9] A. Kirillov, K. He, R. Girshick, C. Rother och P. Dollár, ”Panoptic segmentation,” i *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, s. 9404–9413.

[10] J. Cheng, H. Li, D. Li, S. Hua och V. S. Sheng, ”A survey on image semantic segmentation using deep learning techniques,” 2023.

[11] ”A survey on deep learning techniques for image and video semantic segmentation,” *Applied Soft Computing*, årg. 70, s. 41–65, 2018, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2018.05.018. URL: https://www.sciencedirect.com/science/article/pii/S1568494618302813.

[12] M. Singh m. fl., "Comparative Study of Digital Twin Developed in Unity and Gazebo," *Electronics*, årg. 14, s. 276, 2025. DOI: 10.3390/electronics14020276.

[13] M. Quigley, "ROS: an open-source Robot Operating System," i *IEEE International Conference on Robotics and Automation*, 2009. URL: https://api.semanticscholar.org/CorpusID:6324125.

[14] X. Li och L. Xiang, "Photorealistic Robotic Simulation using Unreal Engine 5 for Agricultural Applications," 2024. DOI: 10.48550/arXiv.2405.18551. arXiv: 2405.18551 [cs.RO]. URL: https://arxiv.org/abs/2405.18551.

[15] S. Macenski, T. Foote, B. Gerkey, C. Lalancette och W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, årg. 7, nr 66, 2022, Hä,tad: 2026-05-05. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/10.1126/scirobotics.abm6074.

[16] G. Metta, P. Fitzpatrick och L. Natale, "YARP: Yet Another Robot Platform," i *International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006. URL: https://www.yarp.it/.

[17] "Welcome to rclUE's documentation! — rclUE 0.1 documentation," hämtad 10 febr. 2026. URL: https://rclue.readthedocs.io/en/latest/.

[18] Tempo Simulation, LLC. "Tempo: simuleringsfokuserade tillägg för Unreal Engine." GitHub-repositorium. URL: https://github.com/tempo-sim/Tempo.

[19] B. Kerbl, G. Kopanas, T. Leimkühler och G. Drettakis, *3D Gaussian Splatting for Real-Time Radiance Field Rendering*, 2023. arXiv: 2308.04079 [cs.GR]. URL: https://arxiv.org/abs/2308.04079.

[20] L. A. K. Irwin, N. C. Coops, K. Anders, G. Mandlburger och L. Winiwarter, "Light detection and ranging of natural systems," *Nature Reviews Methods Primers*, årg. 5, nr 76, 2025. DOI: 10.1038/s43586-025-00446-3. URL: https://doi.org/10.1038/s43586-025-00446-3.

[21] A. Mumuni och F. Mumuni, "Data augmentation: A comprehensive survey of modern approaches," *Array*, årg. 16, s. 100 258, 2022, ISSN: 2590-0056. DOI: https://doi.org/10.1016/j.array.2022.100258. URL: https://www.sciencedirect.com/science/article/pii/S2590005622000911.

[22] C. Shorten och T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, årg. 6, nr 1, s. 1–48, 2019.

[23] R. Singh och N. Sharma, "Improving Brain Tumor Detection and Segmentation with DeepLabV3 and ResNet-50," i *2024 5th IEEE Global Conference for Advancement in Technology (GCAT)*, IEEE, 2024, s. 1–6.

[24] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez och P. Luo, "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers," i *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang och J. W. Vaughan, utg., vol. 34, Curran Associates, Inc., 2021, s. 12 077–12 090. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf.

[25] X. Li m. fl., "Transformer-Based Visual Segmentation: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, årg. 46, nr 12, s. 10 138–10 163, 2024. DOI: 10.1109/TPAMI.2024.3434373.

[26] T. Lin, Y. Wang, X. Liu och X. Qiu, "A survey of transformers," *AI Open*, årg. 3, s. 111–132, 2022, ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2022.10.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651022000146.

[27] K. Weiss, T. M. Khoshgoftaar och D. Wang, "A survey of transfer learning," *Journal of Big data*, årg. 3, nr 1, s. 9, 2016.

[28] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso och A. Torralba, "Scene parsing through ade20k dataset," i *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, s. 633–641.

[29] B. Zhou m. fl., "Semantic understanding of scenes through the ade20k dataset," *International journal of computer vision*, årg. 127, nr 3, s. 302–321, 2019.

[30] R. Azad m. fl., "Loss functions in the era of semantic segmentation: A survey and outlook," *arXiv preprint arXiv:2312.05391*, 2023.

[31] M. Yeung, E. Sala, C.-B. Schönlieb och L. Rundo, "Unified Focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation," *Computerized Medical Imaging and Graphics*, årg. 95, s. 102 026, 2022, ISSN: 0895-6111. DOI: https://doi.org/10.1016/j.compmedimag.2021.102026. URL: https://www.sciencedirect.com/science/article/pii/S0895611121001750.

[32] R. O. Ogundokun, R. Maskeliunas, S. Misra och R. Damaševičius, "Improved CNN based on batch normalization and adam optimizer," i *International conference on computational science and its applications*, Springer, 2022, s. 593–604.

[33] M. Yaqub m. fl., "State-of-the-art CNN optimizer for brain tumor segmentation in magnetic resonance images," *Brain sciences*, årg. 10, nr 7, s. 427, 2020.

[34] Z. Li och D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, årg. 40, nr 12, s. 2935–2947, 2017.

[35] R. Varghese och S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness," i *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 2024, s. 1–6. DOI: 10.1109/ADICS58448.2024.10533619.

[36] T.-Y. Lin m. fl., "Microsoft coco: Common objects in context," i *European conference on computer vision*, Springer, 2014, s. 740–755.

[37] P. Trentsios, M. Wolf och D. Gerhard, "Overcoming the Sim-to-Real Gap in Autonomous Robots," *Procedia CIRP*, årg. 109, s. 287–292, 2022, 32nd CIRP Design Conference (CIRP Design 2022) - Design in a changing world, ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2022.05.251. URL: https://www.sciencedirect.com/science/article/pii/S2212827122007004.

[38] M. Chaman, A. E. Maliki, H. Dahou och A. Hadjoudja, "Benchmarking YOLO-based deep learning models for real-time object detection in hybrid ADAS and intelligent transportation systems," *Results in Engineering*, årg. 29, s. 108 942, 2026, ISSN: 2590-1230. DOI: https://doi.org/10.1016/j.rineng.2025.108942. URL: https://www.sciencedirect.com/science/article/pii/S2590123025049849.

[39] W. Gu, S. Bai och L. Kong, ”A review on 2D instance segmentation based on deep neural networks,” *Image and Vision Computing*, årg. 120, s. 104 401, 2022, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2022.104401. URL: https://www.sciencedirect.com/science/article/pii/S0262885622000300.

[40] N. Jegham, C. Y. Koh, M. Abdelatti och A. Hendawi, ”Yolo evolution: A comprehensive benchmark and architectural review of yolov12, yolo11, and their previous versions,” *arXiv preprint arXiv:2411.00201*, 2024.

[41] P. S. S. Markappa, C. O’Leary och C. Lynch, ”A Review of YOLO Models for Soccer-Based Object Detection,” i *2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS)*, 2024, s. 1–7. DOI: 10.1109/ICDS62089.2024.10756443.

[42] E. Brorsson, ”Camera-Based Perception under Domain Shifts: Efficient Training and Multi-View Fusion for Mobile Robots in Internal Logistics,” Hämtad 2026-05-06, Licentiatavhandling, Institutionen för elektroteknik, system och reglerteknik, Göteborg, Sverige, 2025. URL: https://research.chalmers.se/en/publication/548217.

[43] Appoose och P. Rosengren, *Homography translation illustration*, https://commons.wikimedia.org/wiki/File:Homography-transl.svg, Licens: CC BY 3.0. Hämtad 2026-05-06, 2011.

[44] VirtualMich. ”Lär dig OpenCV.” Hämtad 4 maj 2026, hämtad 4 maj 2026. URL: https://virtualmich.com/sv/l%C3%A4r-dig-opencv/.

[45] J. Held m. fl., *MeshSplatting: Differentiable Rendering with Opaque Meshes*, 2025. arXiv: 2512.06818 [cs.CV]. URL: https://arxiv.org/abs/2512.06818.

[46] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li och Y. Gao, ”A survey on federated learning,” *Knowledge-Based Systems*, årg. 216, s. 106 775, 2021, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.106775. URL: https://www.sciencedirect.com/science/article/pii/S0950705121000381.

[47] Q.-C. Pham, R. Madhavan, L. Righetti, W. Smart och R. Chatila, ”The Impact of Robotics and Automation on Working Conditions and Employment [Ethical, Legal, and Societal Issues],” *IEEE Robotics & Automation Magazine*, årg. 25, nr 2, s. 126–128, 2018. DOI: 10.1109/MRA.2018.2822058.

[48] Z. Hosseini, S. Nyholm och P. M. Le Blanc, ”The Ethics of Developing, Implementing, and Using Advanced Warehouse Technologies: Top-Down Principles Versus The Guidance Ethics Approach,” *Journal of Human-Technology Relations*, årg. 2, s. 1–25, 2024.

A

Appendix 1

A.1 Hårdvara

Xgrids LiDAR specifikationer[1]:

- Laser klass: klass 1 / 940 nm
- Skanningsområde: 0.1 m - 30 m vid 10% reflektans, upp till 60 m vid 90% reflektans
- Synfält: 180deg x 180deg
- Infångningsgrad 856 000 punkter/sekund

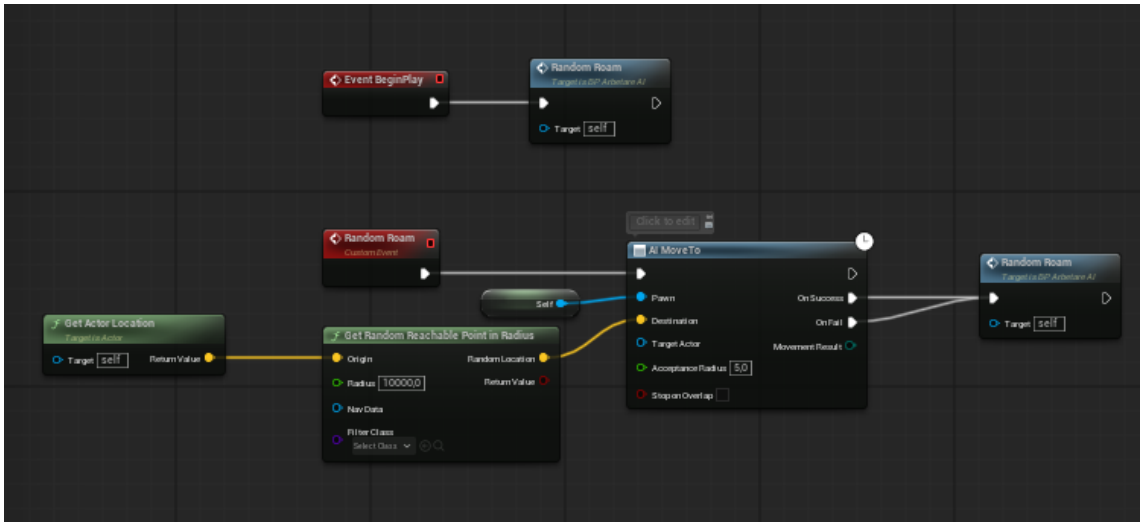
Dator för rendering av 3D Gaussian Splatt:

- Grafikkort: NVIDIA RTX 5090
- RAM-minne: 128 GB

Dator för 3D-modulering:

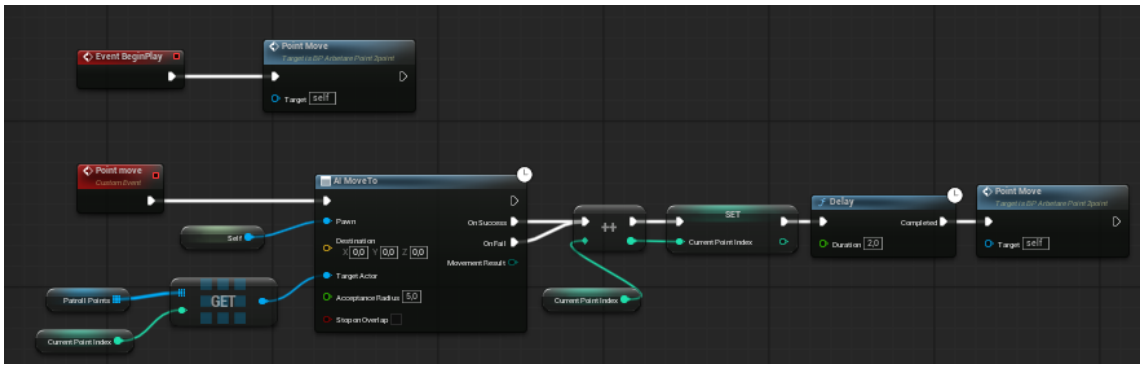
- Grafikkort: NVIDIA RTX 3080
- RAM-minne: 32 GB

A.2 Blueprints

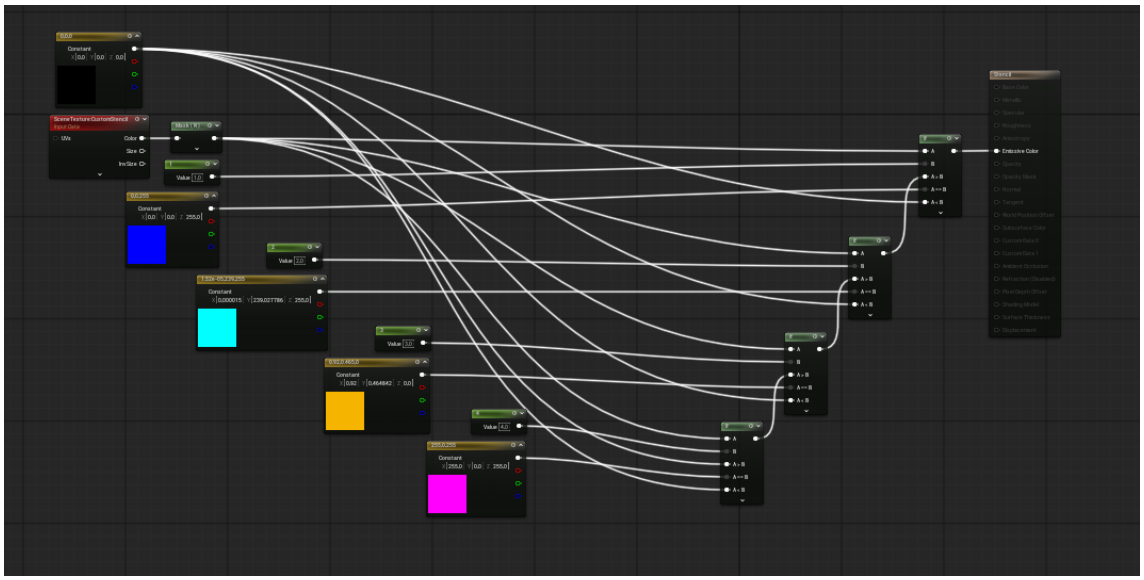


Figur A.1: Blueprint för slumpartad navigering.

[1]https://www.laserscanning-europe.com/en/mobile-laser-scanners/xgrids-portalcam

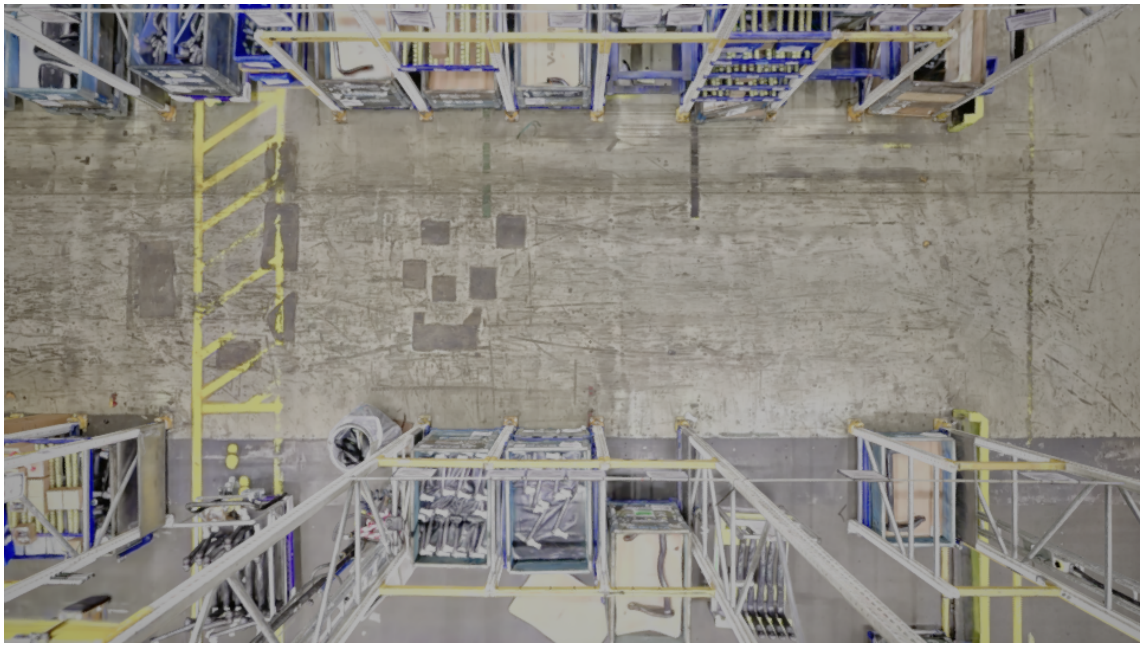


Figur A.2: Blueprint för styrning med sekventiella hållpunkter.



Figur A.3: Blueprint för konvertering av stencilvärden från gråskala till färger. ID 1 motsvarar färgen blå, ID 2 motsvarar färgen turkos, ID 3 motsvarar färgen orange och ID 4 motsvarar färgen magenta.

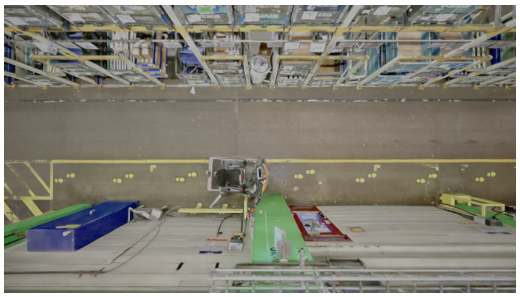
A.3 Bild med låg mättnad jämfört med normalt mättad bild vid genereringen av datasamlingen



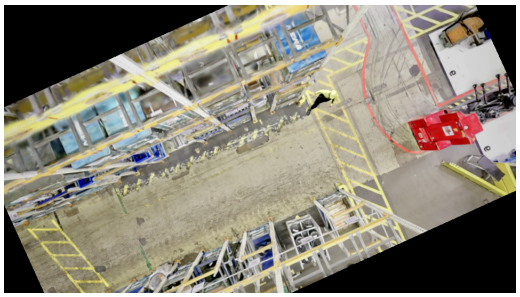
Figur A.4: Genererad bild från simuleringsmiljön med 15-20 % lägre mättnad.

Figur A.5: Genererad bild från simuleringsmiljön med normal mättnad.

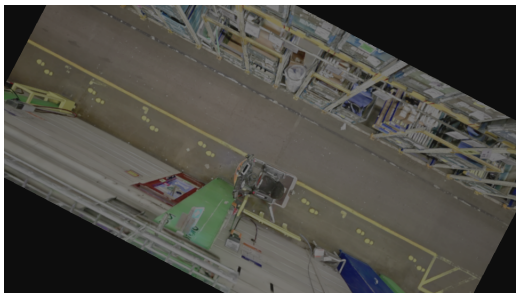
A.4 Olika sorters datautökning



(a) Bild utan justering från simuleringsmiljön.



(b) Bild med ökad ljusstyrka samt rotation från simuleringsmiljön.

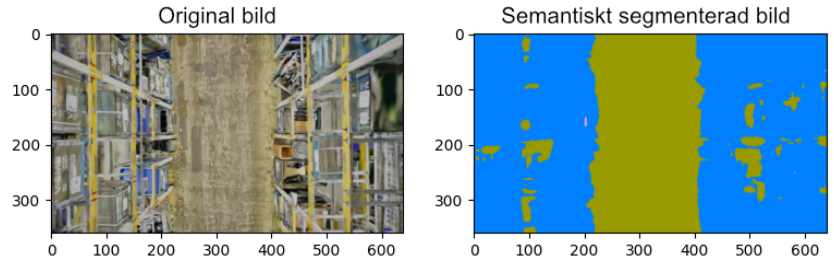


(c) Bild med ökad kontrast samt rotation från simuleringsmiljön.

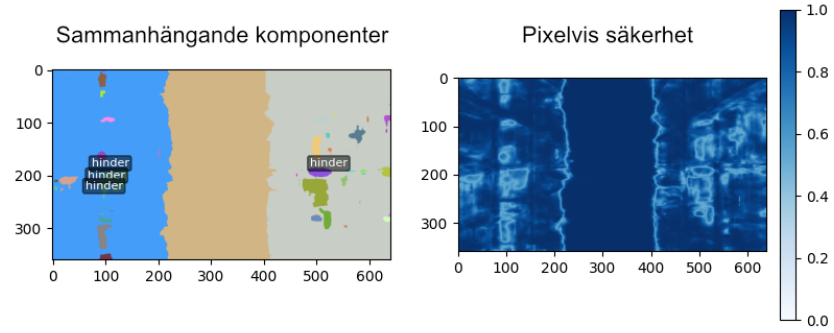
(d) En roterad horisontellt speglad bild från simuleringsmiljön.

Figur A.6: Exempel på dataökning i simuleringsmiljön.

A.5 Semantiskt segmenterade bilder med SegFormer-B0.

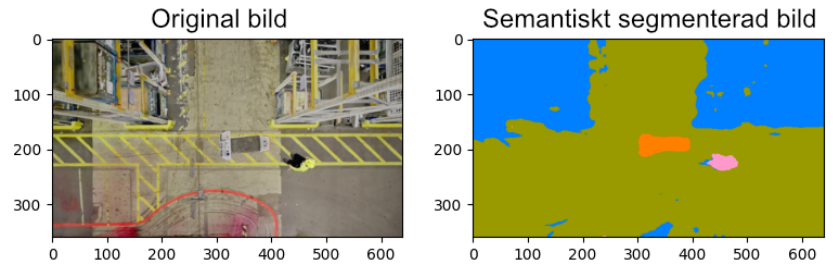


(a)

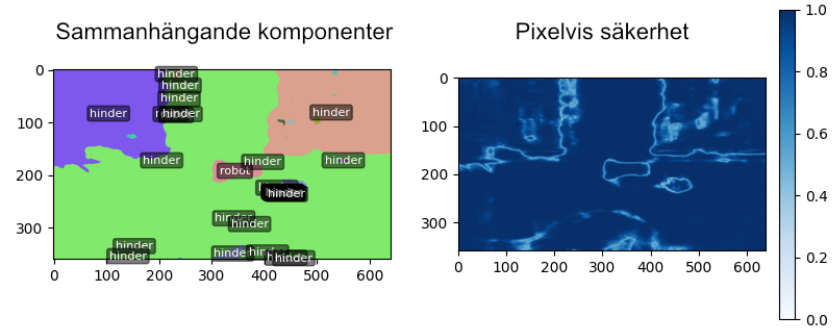


(b)

Figur A.7: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

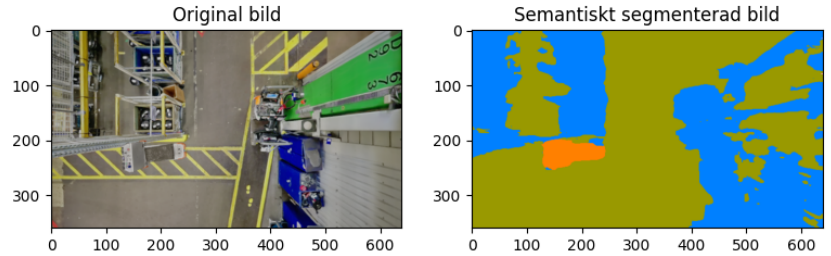


(a)

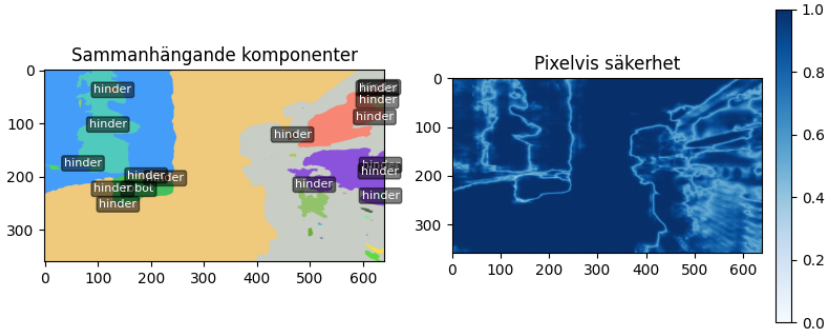


(b)

Figur A.8: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

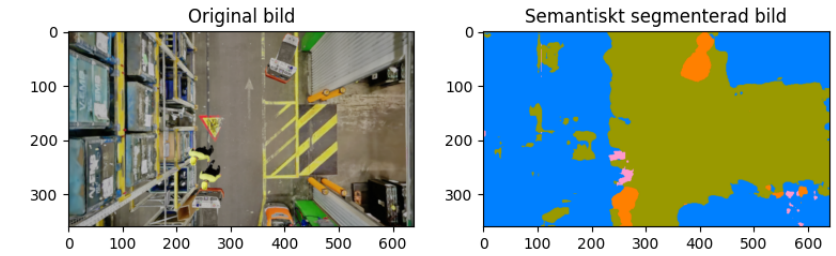


(a)

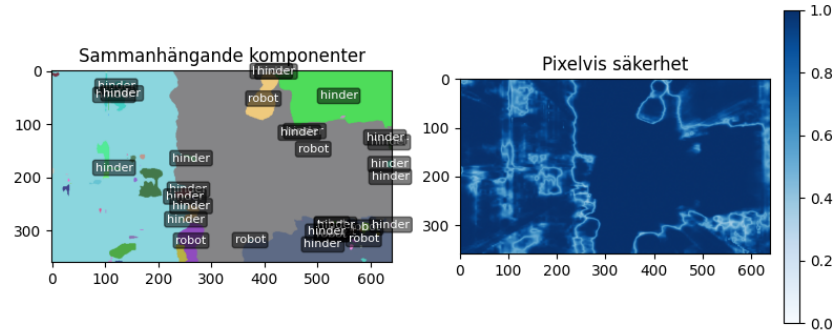


(b)

Figur A.9: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

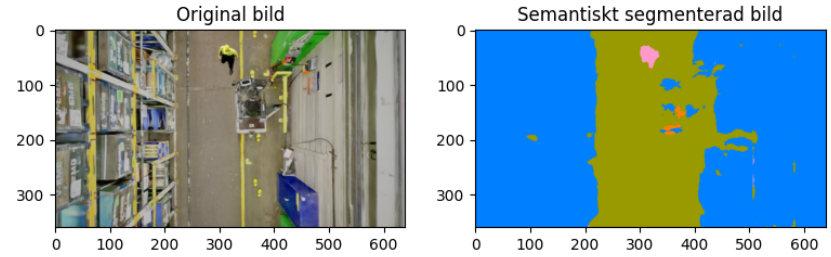


(a)

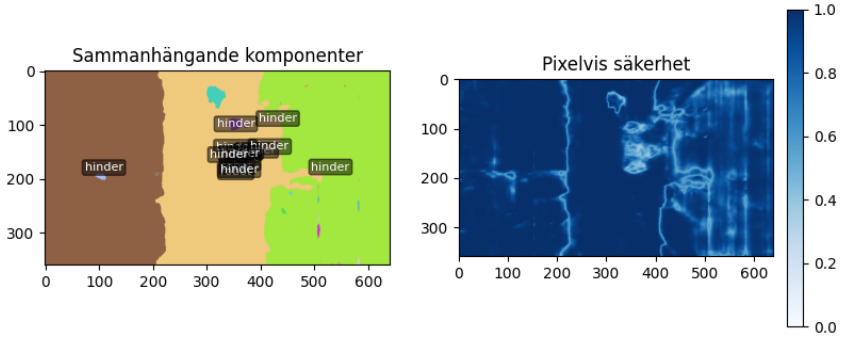


(b)

Figur A.10: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

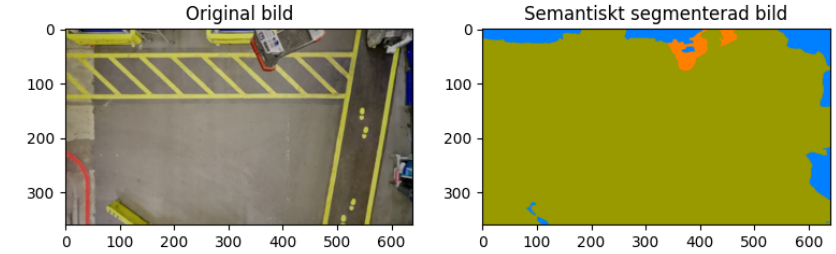


(a)

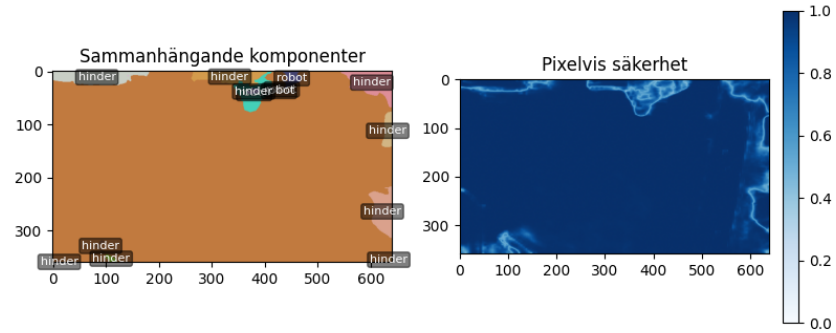


(b)

Figur A.11: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

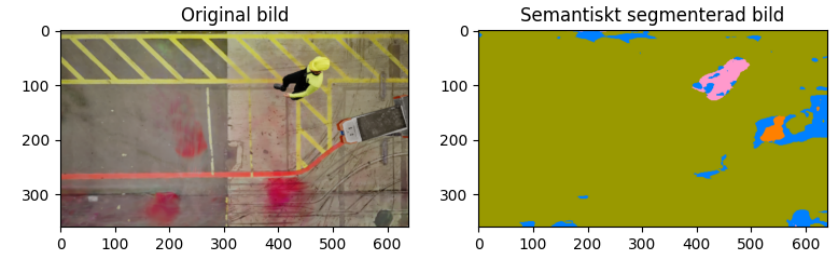


(a)

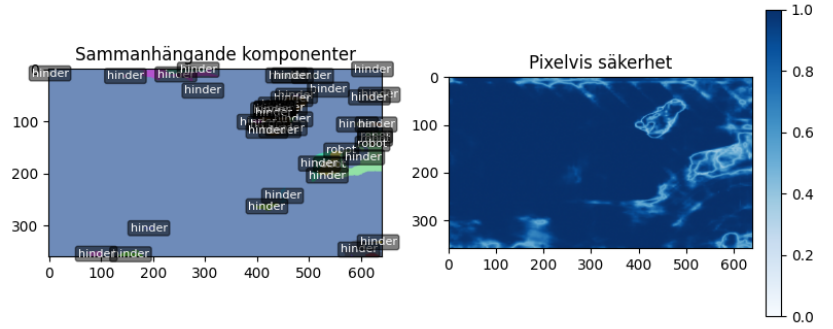


(b)

Figur A.12: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.



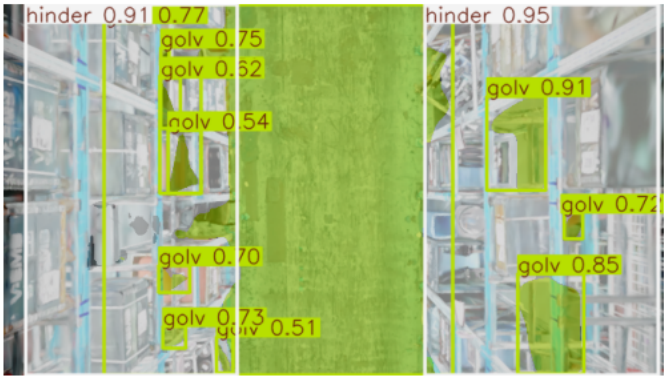
(a)



(b)

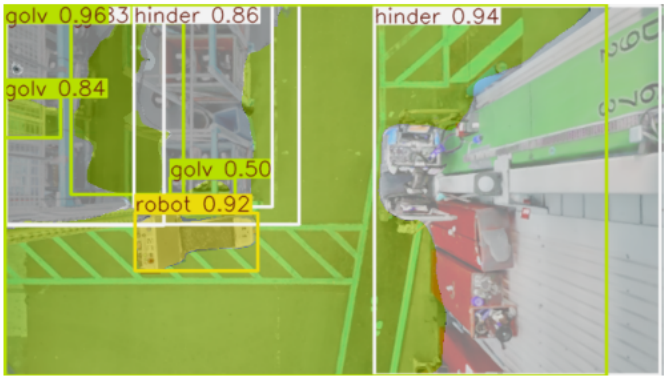
Figur A.13: Semantiskt segmenterade bilder med SegFormer-B0, där (a) innehåller original bild och den semantiskt segmenterade bilden, och (b) innehåller figurer på de extraherade instanserna och den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

A.6 Semantisk segmenterade bilder med YOLOv8-Seg Nano

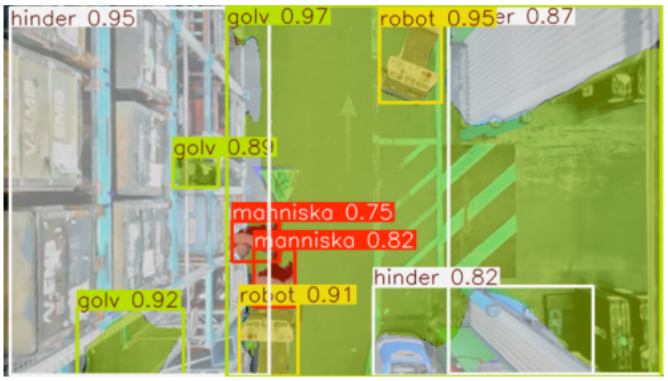


Figur A.14: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

Figur A.15: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

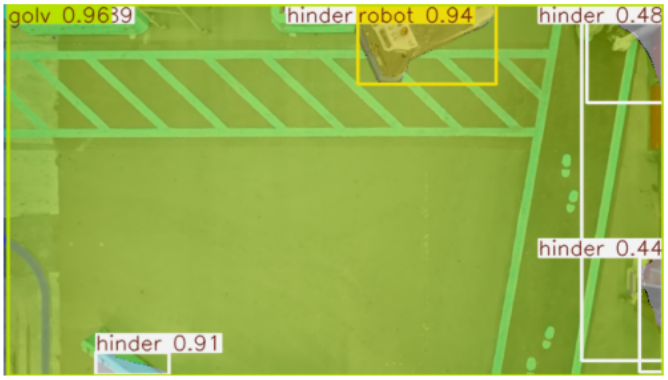


Figur A.16: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

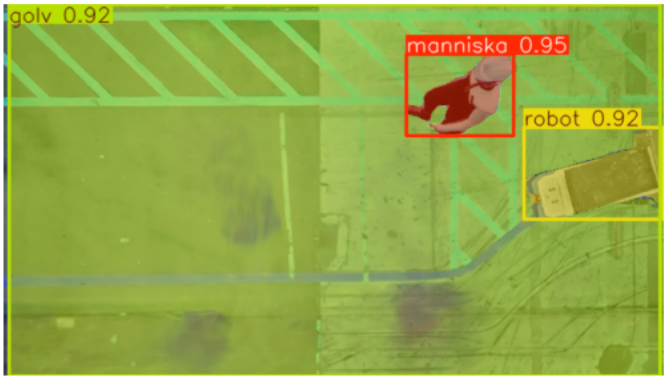


Figur A.17: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

Figur A.18: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

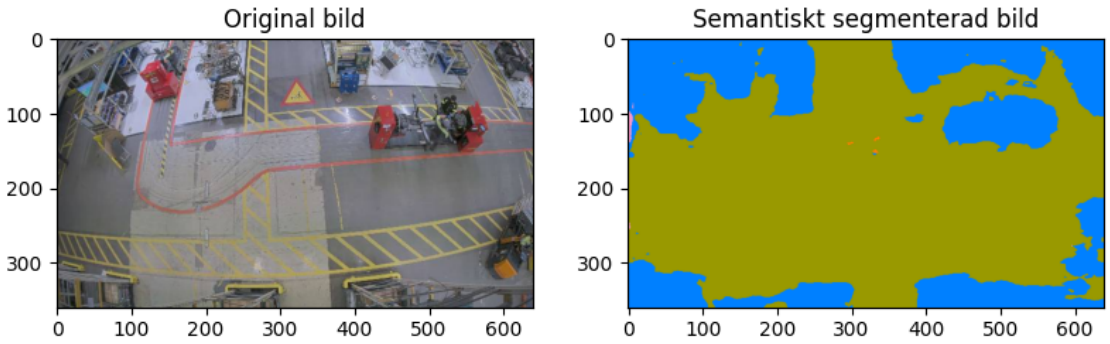


Figur A.19: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

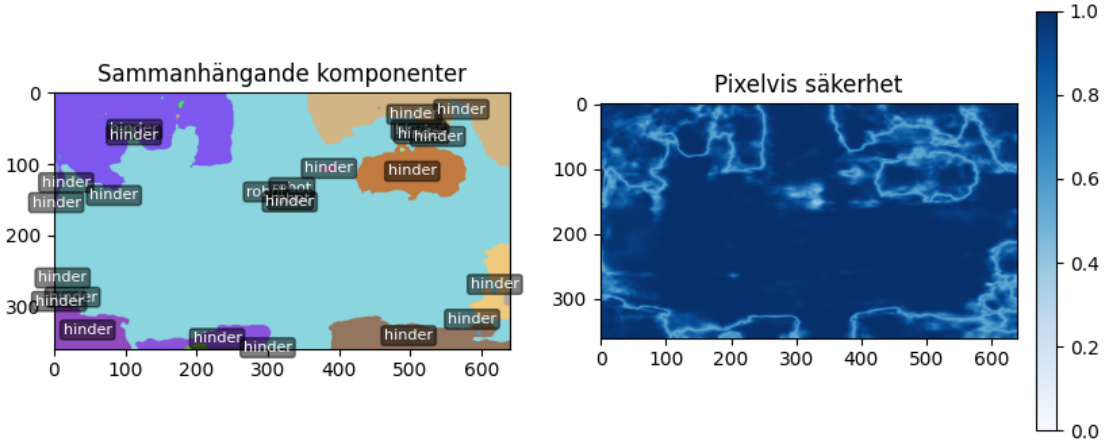


Figur A.20: Instans segmenterad bild med YOLOv8-Seg Nano som innehåller den segmenterade bilden ovanpå originalbilden. Säkerheten för varje identifiering visas även som en etikett.

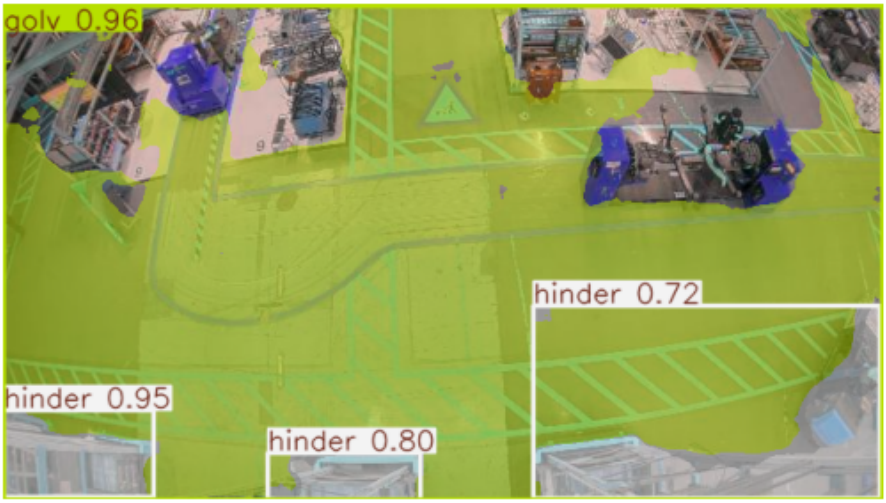
A.7 Segmentering av bilder från verkligheten med SegFormer-B0 och YOLOv8-Seg Nano



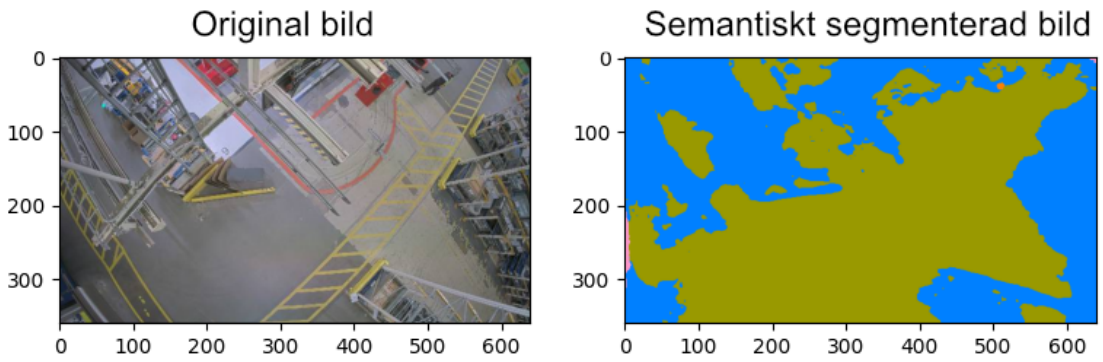
Figur A.21: Original bild från verkligheten till vänster i figuren och semantiskt segmenterad bild till höger med hjälp av den tränade SegFormer-B0 modellen.



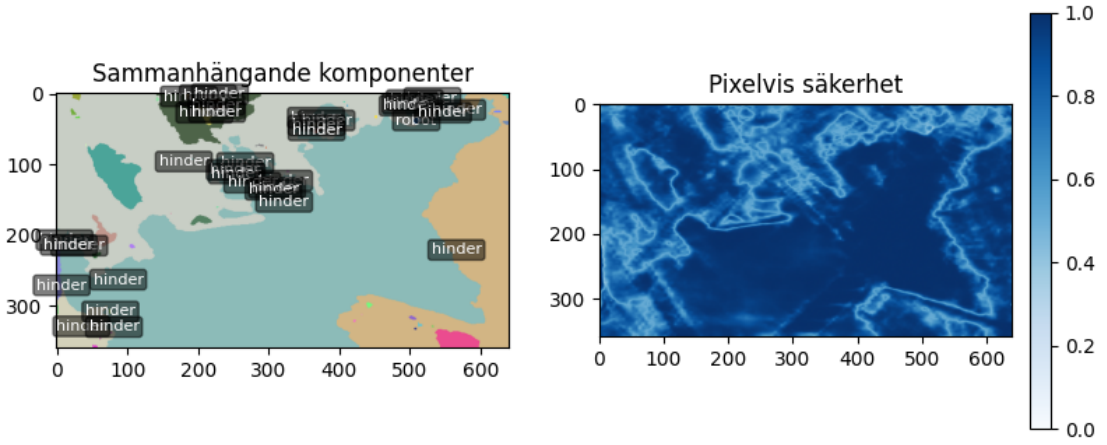
Figur A.22: Sammanhängande komponenter till vänster som visar dom extraherade instanserna, till höger visas den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.



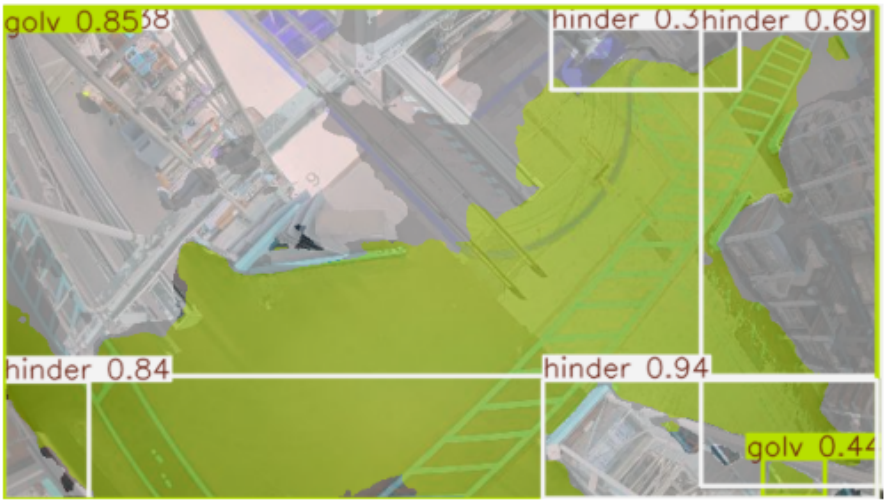
Figur A.23: Instans segmenterad bild med hjälp av den tränade YOLOv8-Seg Nano modellen.



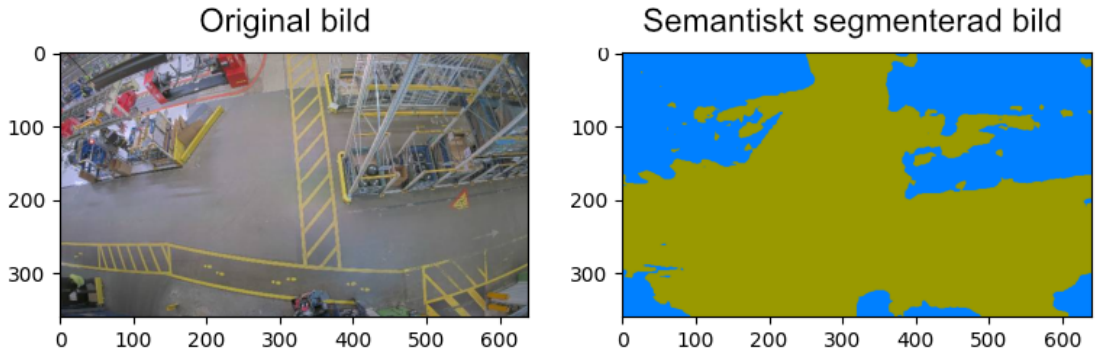
Figur A.24: Original bild från verkligheten till vänster i figuren och semantiskt segmenterad bild till höger med hjälp av den tränade SegFormer-B0 modellen.



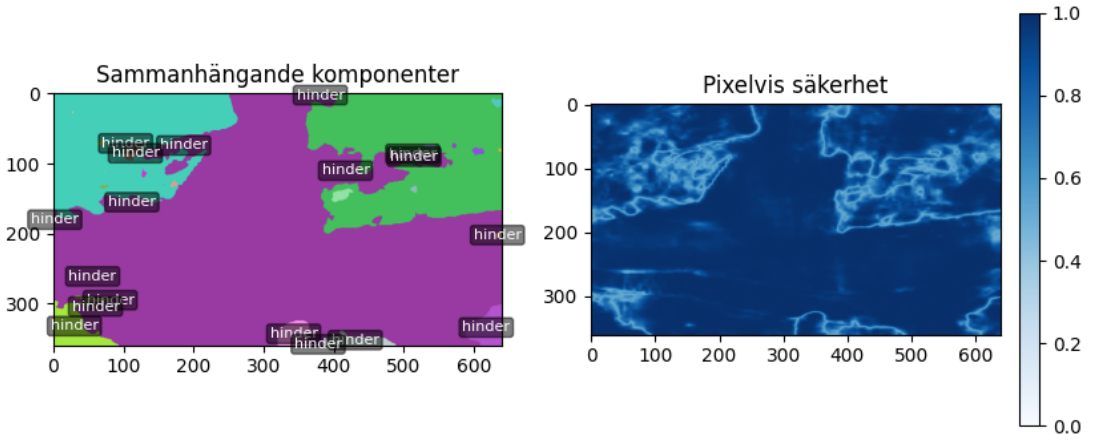
Figur A.25: Semantisk segmentering med SegFormer-B0 på verkliga bilder. Sammanhängande komponenter till vänster som visar dom extraherade instanserna, till höger visas den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.



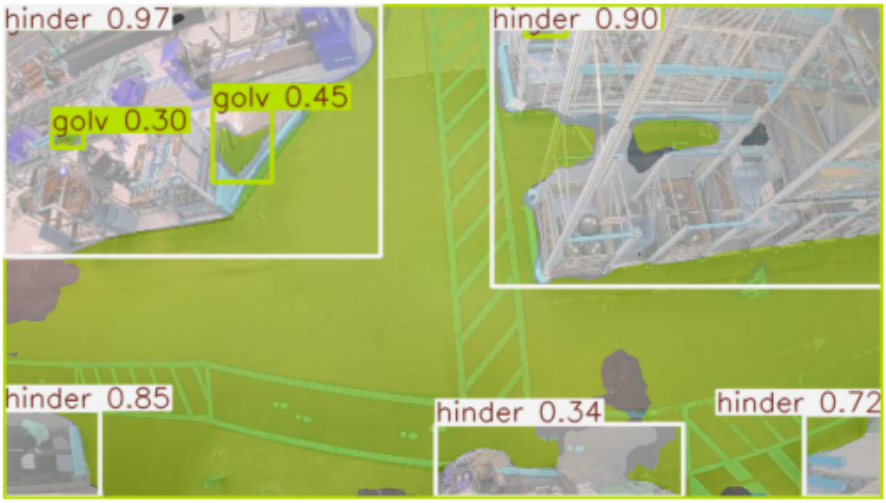
Figur A.26: Instans segmenterad bild från verkligheten med hjälp av den tränade YOLOv8-Seg Nano modellen.



Figur A.27: Original bild från verkligheten till vänster i figuren och semantiskt segmenterad bild till höger med hjälp av den tränade SegFormer-B0 modellen.

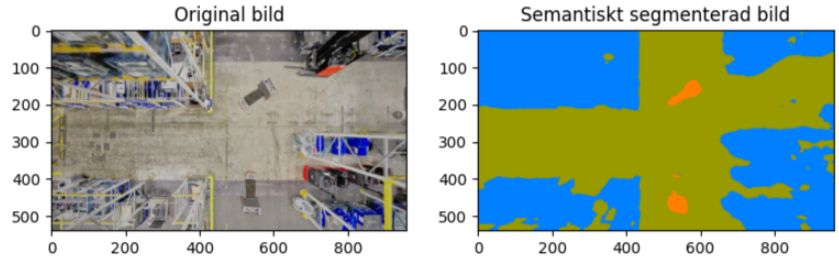


Figur A.28: Semantisk segmentering med SegFormer-B0 på verkliga bilder. Sammanhängande komponenter till vänster som visar dom extraherade instanserna, till höger visas den pixelvisa säkerheten, där blå är hög säkerhet och vit är låg säkerhet.

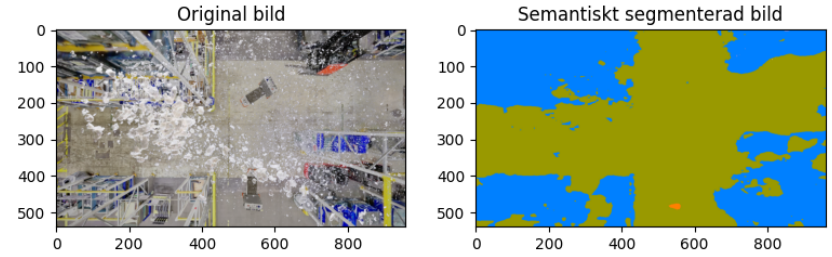


Figur A.29: Instans segmenterad bild från verkligheten med hjälp av den tränade YOLOv8-Seg Nano modellen.

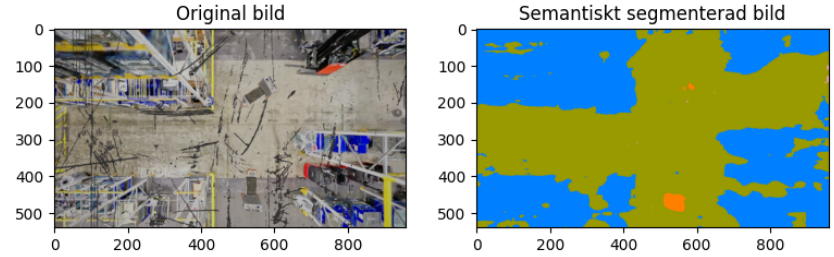
A.8 Segmentering på ovanliga händelser med kameror



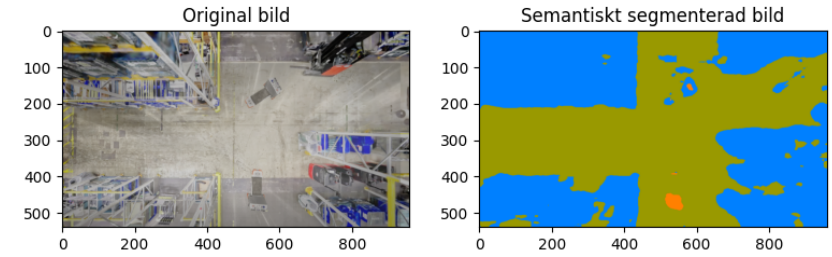
(a)



(b)

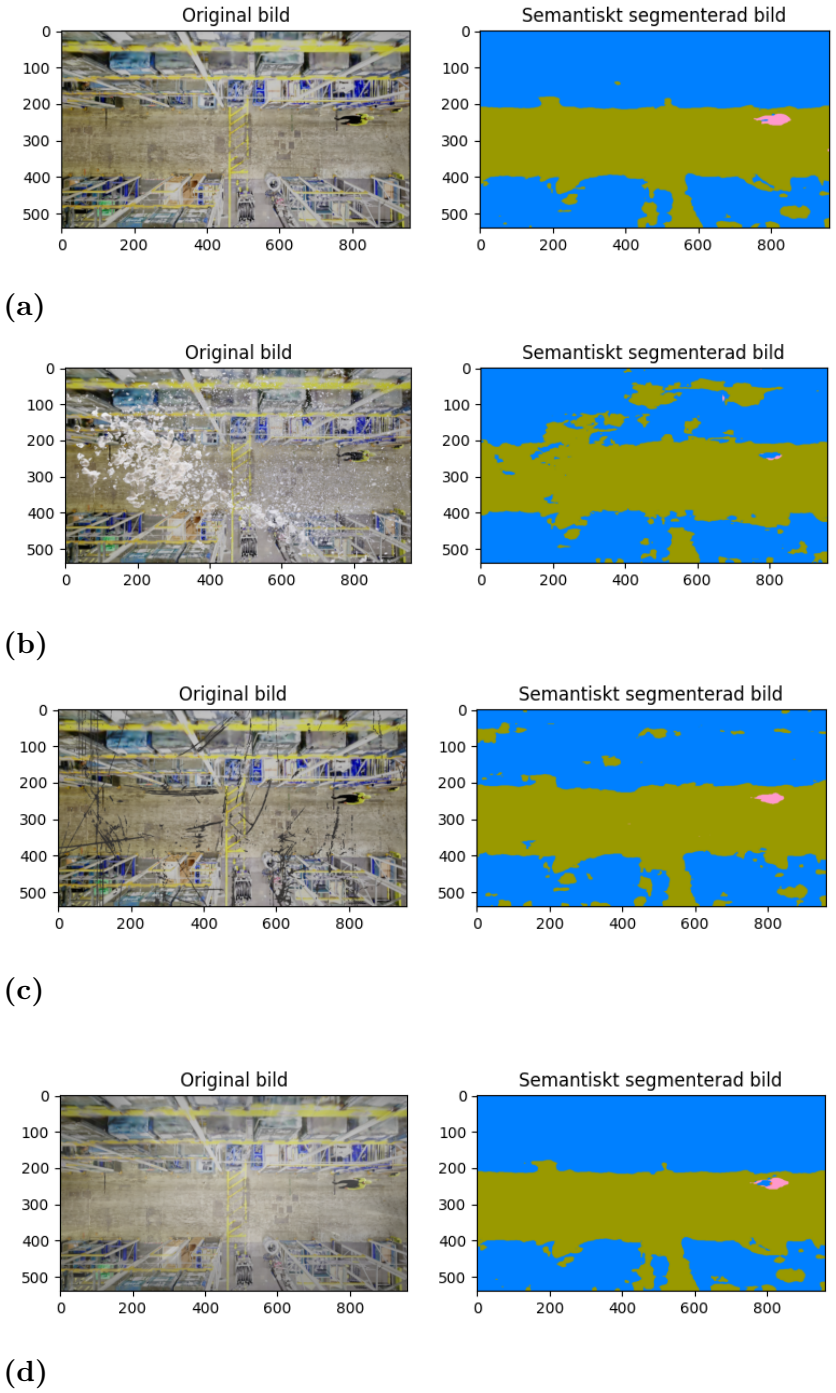


(c)



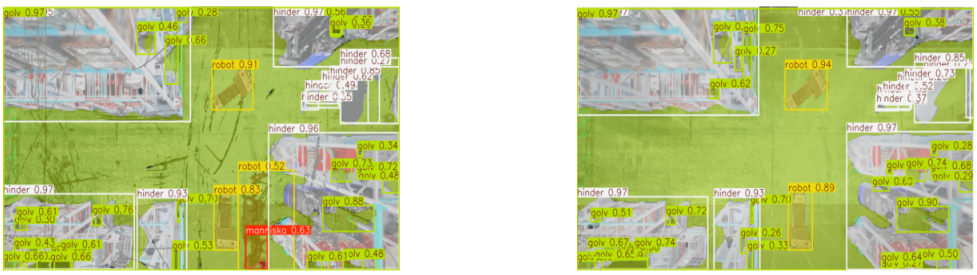
(d)

Figur A.30: Jämförelse på den semantiska segmenteringen med SegFormer-B0 på olika filter för att representera ovanliga händelser (a) Utan filter. (b) Med ett smuts filter. (c) Filter med repor, och (d) Filter med ljusreflektion.

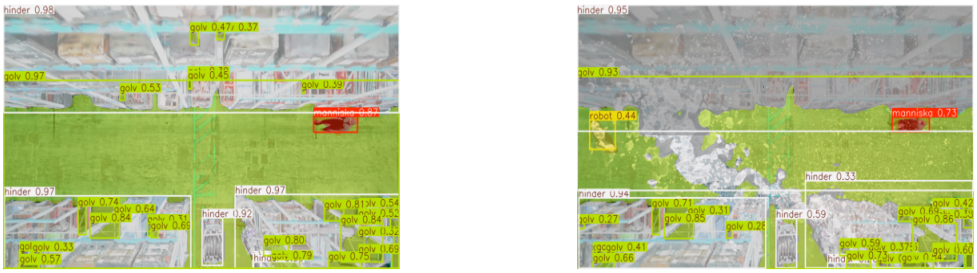


Figur A.31: Jämförelse på den semantiska segmenteringen med SegFormer-B0 på olika filter för att representera ovanliga händelser med en bild från en annan kameravinkel i där bild:(a) Utan filter, (b) Med ett smuts filter, (c) Filter med repor, och (d) Filter med ljusreflektion.

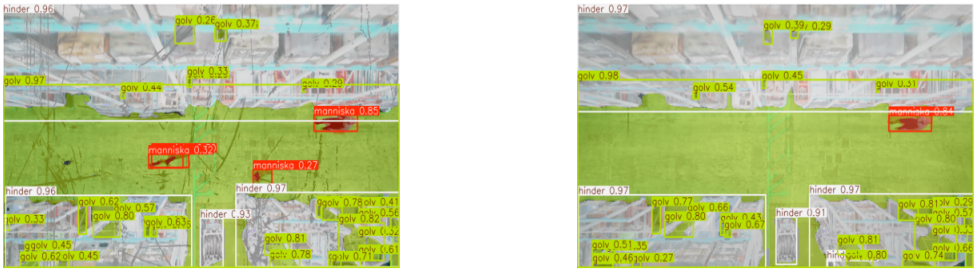
Figur A.32: Jämförelse på instans segmenteringen med YOLOv8-Seg Nano på olika filter för att representera ovanliga händelser med en bild från en annan kameravinkel i där vänstra bilden är utan filter och högra bilden är med ett smutsfilter.



Figur A.33: Jämförelse på instans segmenteringen med YOLOv8-Seg Nano på olika filter för att representera ovanliga händelser med en bild från en annan kameravinkel i där vänstra bilden filter med repor och högra bilden är filter med ljusreflektion.



Figur A.34: Jämförelse på instans segmenteringen med YOLOv8-Seg Nano på olika filter för att representera ovanliga händelser med en bild från en annan kameravinkel i där vänstra bilden är utan filter och högra bilden är med ett smutsfilter.



Figur A.35: Jämförelse på instans segmenteringen med YOLOv8-Seg Nano på olika filter för att representera ovanliga händelser med en bild från en annan kameravinkel i där vänstra bilden filter med repor och högra bilden är filter med ljusreflektion.

Institutionen för Elektroteknik
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS