Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception

Ashwat Chidambaram

Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception

by Ashwat Chidambaram

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Murat Arcak

Professor Murat Arcak
Research Advisor

May 10, 2024

Date

* * * * * * *

[illegible]

Professor Allen Y. Yang
Second Reader

May 10, 2024

Date

Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception

by

Ashwat Chidambaram

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Murat Arcak, Chair
Professor Allen Y. Yang, Co-chair

Spring 2024

Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception

Abstract

Leveraging Zero-Shot Sim2Real Learning to Improve Autonomous Vehicle Perception

by

Ashwat Chidambaram

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Murat Arcak, Chair

Professor Allen Y. Yang, Co-chair

Autonomous vehicles are one of the most exciting technologies at the forefront of innovation in the current era we live in. In particular, accurate perception of the world around the vehicle is a key component for safe and reliable autonomy. As a result, improving autonomous vehicle perception models to achieve peak performance is an ideal goal that has been an active area of research, oftentimes bottlenecked by the necessity for high quality training data. The use of simulation engines to generate data typically aims to account for this data scarcity, but unfortunately the outputs often look visually dissimilar to the real-world domain of images, a problem that is commonly referred to as the "reality gap". Through the work presented in this thesis, we aim to bridge this reality gap through the use of zero-shot sim2real learning approaches, in order to generate realistic training data for perception models with limited real-world baseline images to begin with. In our specific context for this project, we demonstrate significantly improved performance for an autonomous race car in high-speed races against other cars on a professional track, with an enhanced ability to detect and segment these opponent vehicles in diverse scenarios compared to before. Furthermore, our work ultimately demonstrates a versatile approach to pipeline simulation data into sim2real outputs for training real-world models, unlocking a new level of data generation that allows us to create practically infinite scenarios, and ultimately improve the robustness of autonomous vehicle perception in novel unseen domains.

To my beloved father and mother, Chidambaram Kandasamy and Rajam Chidambaram,
and my amazing brother, Ashwin Chidambaram, this one's for you.

Contents

Acknowledgments

My journey wouldn't have been possible without the invaluable support of so many incredible people that I am grateful to have had in my life.

First and foremost, I would like to express my heartfelt gratitude to my research advisor Prof. Murat Arcak and my research mentor Prof. Allen Y. Yang, for providing me with this amazing research opportunity and invaluable support throughout my Masters.

I would also love to thank Chris Lai, Wayne Lai, Eric Berndt, C.K. Wolfe, and the rest of the AI Racing Tech team for their contributions to my success throughout the project. Their expertise and guidance have been paramount in helping me advance my research, and ultimately achieving success through this project for the Indy Autonomous Challenge.

Beyond my academic mentors and colleagues, I would also like to sincerely express my appreciation for all the friends who supported me throughout my five years at UC Berkeley. Thank you for making college an amazing experience, and for the priceless unforgettable memories we made along the way.

To the countless other individuals who have played an integral part in my life, every piece of advice, every helping hand, and every moment of support, no matter how big or small, has shaped my journey and made me who I am today.

Finally, I am forever grateful for my dad, mom, and brother, who have always encouraged me to shoot for the stars and become the best version of myself every single day. Words can never describe how extremely fortunate I am to have such a loving and supportive family, and the reason I get to call myself a proud Golden Bear for the rest of my life.

Thank you to everyone who played a part in my success and growth. I hope to pass on all your kindness and encouragement, and inspire countless future students and engineers to strive for excellence, the same way I felt inspired and supported by all of you.

From the bottom of my heart once again, thank you. I couldn't have done it without you all. Here's to an exciting journey ahead, and to the storytellers of a future yet unwritten.

–AC

Chapter 1

Introduction

1.1 Competition Overview

The field of perception for autonomous vehicles has been explored for many years. From the initial days of the DARPA Grand Challenge, bringing together the brightest students from the best universities around the United States, the goal was to create autonomous unmanned vehicles to navigate hundreds of miles of desert terrain. This challenge gave birth to a whole industry of autonomous vehicles, from which many prominent startups were spun off and further developed. These companies became some of the pioneers of self-driving technology, and are leading the industry today at the forefront of innovation in this field.

Recently, the concept of autonomy delved further into other aspects of driving, particularly into the exhilarating thrill of racing and motorsport. The Indy Autonomous Challenge (IAC) is organized by the Energy Systems Network (ESN), which is an Indianapolis-based non-profit and initiative of the Central Indiana Corporate Partnership (CICP) and CICP Foundation, Inc. Since its creation in late 2019, it grew to become a massive network of public-private partnerships working alongside academic institutions to create the world's first fully autonomous race cars. The automotive platform that the IAC uses is the Dallara AV-21/AV-24 series of vehicles, pictured below.

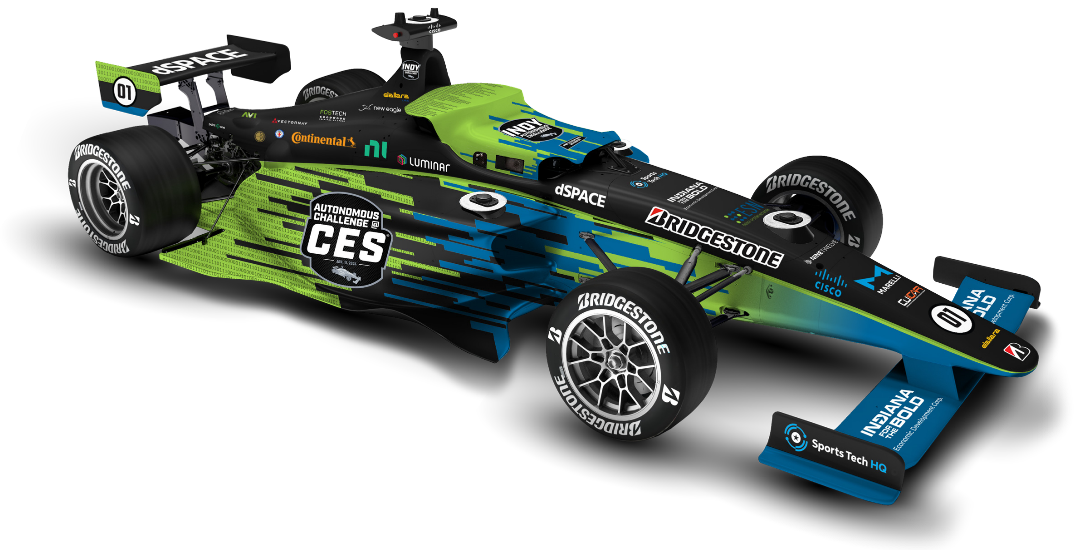


Figure 1.1: Indy Autonomous Challenge Dallara AV-21 Vehicle

Every year, various teams consisting of smaller groups of universities compete in different races at tracks all around the world. These includes the Las Vegas Motor Speedway as the Consumer Electronics Show (CES), the Indianapolis Motor Speedway in Indiana, the Autodromo Nazionale Monza (Monza Circuit) in the city of Milan in northern Italy, and many more.

As stated on the official competition website in 2024, the primary goal of the Indy Autonomous Challenge (IAC) is "to advance technology that can speed the commercialization of fully autonomous vehicles and deployments of advanced driver-assistance systems (ADAS). These enhancements will lead to increased safety and performance in all modes of motorsports and commercial transportation" [7]. In addition to the above, the competition also serves as an amazing platform for students to develop their skills and excel in the field of Science, Technology, Engineering, and Mathematics (STEM).

1.2 Team Overview

Our team, AI Racing Tech (ART), is one among many others in this competition. The team is a joint partnership between the UC Berkeley College of Engineering, UC San Diego, Carnegie Mellon University, and the University of Hawai‘i. Each university works across different aspects of the tech stack, and UC Berkeley is the lead behind the entire perception stack for our racing vehicle. This is where our team creates and innovates the next generation of perception algorithms tailored for high-speed racing, with implications that can reach farther into the realm of autonomous vehicle perception as a whole.

1.3 Problem Statement

The goal of this work is to find novel ways in order to augment our racecar to be prepared for future races in unseen domains, whether that be on new tracks, with multiple vehicles, in dangerous edge case scenarios, etc. We need to tackle this problem from the perspective of computer vision, in particular regarding high-speed 2D vision on the race track. As we will go over in future sections, our data is currently very limited. However, the goal of the competition is to race with multiple race cars on the track at once, much like traditional motor sports such as NASCAR or Formula 1 racing. Hence, we must be able to train our autonomous race car to be capable of racing on the track alongside other vehicles, despite having little to no prior data or experience doing so due to the novelty of the competition and limited resources.

To solve this problem, all teams currently leverage simulation engines in order to generate practically limitless examples of racing scenarios. All onboard sensors and cameras are simulated as well through the use of a digital twin, in order to utilize the simulation training to apply to real world vehicles thereafter. However, there is often a clear difference between simulation and reality, often called the ”reality gap” which is a ubiquitous problem in the field of robotics and autonomy. In particular to the application of computer vision for onboard cameras, the visual differences between simulation engines and real-world camera images are stark, which will be discussed in upcoming sections.

Hence, our problem statement can clearly be defined as follows. Given limited inputs of single-vehicle real-world images and plentiful simulation images containing multiple vehicles in view, we aim to produce the most realistic output data for training our models by generating sim2real images. Since our sim2real goal intends to fill the gap due to a lack of sufficiently large real-image training datasets for our problem, we cannot adopt intensive pre-trained models because they would require large amounts of data to begin with. Hence, we aim to utilize a few-shot or zero-shot approach in order to maximize the efficiency of our model and be robust to the constraint of limited real-world baseline data.

Ultimately, we must seek a way in order to bridge this reality gap, and minimize the differences between simulation and reality. This fundamental problem is the entire foundation upon which this project was born. Through our work outlined in this paper, we aim to find a solution to close this gap between our current capabilities with ideal future aspirations, ultimately striving towards the goal of winning out other teams in future races hosted by the Indy Autonomous Challenge. May the best team win!

Chapter 2

System Overview

At its core, our autonomous racecar is very similar to a traditional racecar that can be found in the Formula One style of vehicles. The key difference is the lack of a driver's cockpit, instead fitted with a whole suite of sensors and onboard compute. Below is an image of our specific AI Racing Tech's vehicle, which is built off the Dallara AV-21 platform.

Figure 2.1: AI Racing Tech's Dallara AV-21 Competition Vehicle

2.1 Software Stack

The software stack of our vehicle runs primarily on Ubuntu 22.04, using ROS 2 and the Galactic Cyclone DDS middleware. Building off the base Indy Autonomous Challenge (IAC) open source software stack, the AI Racing Tech (ART) software system involves a plethora of moving parts, for the simulation, perception, controls, firmware, and many other subteams.

2.2 Hardware Stack

The hardware stack of our vehicle contains three Luminar Hydra LiDARs, with 64 Lines and 120 degree scanning. It further contains two Novatel GPS units with millimeter accuracy. Regarding cameras, it contains four wide field Mako cameras and two narrow field Mako cameras. There are also three radar sensors. Finally, there is a state-of-the art NVIDIA RTX A6000 GPU integrated with the dSpace x86 processing system onboard.

2.3 Subteam Responsibilities

- **Controls:** This team handles all the low-level hardware control such as steering, braking, gear shifting, and accelerating via software model predictive control algorithms.
- **Localization:** This team works on GPS and IMU fusion, Kalman filtering, pose estimation, and fault and failure handlers.
- **Perception:** This team develops software for sensor ingestion, sensor fusion, filtering, and world-state estimation. They also cover all aspects of object detection, classification, image segmentation, pose estimation, and prediction.
- **Vehicle Dynamics & Simulation:** This team handles race line design and optimization, real-time vehicle dynamics, satellite perception sensor image analysis, the development of simulated vehicle models to test software, and the creation of digital twins.
- **Planning:** This team is in charge of navigation and strategy, high-level vehicle control, and path planning and obstacle avoidance.
- **Trackside Operations:** This team prepares the vehicle for launch, keeps the mechanical systems operational, and essentially acts as mission control for the race vehicle.

The work covered in this paper is under the perception team, which is led by the UC Berkeley College of Engineering. In particular for the scope of this thesis, the relevant hardware and software is a subset of the full pipeline. Regarding hardware, this project focuses on utilizing 2D camera data streamed from the wide-field Mako cameras mounted at the front of the vehicle. The majority of data collection and experimentation specifically involves the front-left center mounted camera. On the software side, all coding is done entirely in Python3, using various libraries such as TensorFlow, PyTorch, OpenCV, NumPy, Keras, Pandas, and other relevant frameworks. The models are trained offline in our research lab at Cory Hall using a computer equipped with the NVIDIA GeForce RTX 3080 GPU.

Chapter 3

Related Works

At its core, the sim2real problem we are aiming to solve can be viewed from a high-level perspective as a style transfer problem, which has been tackled for many years by researchers. A brief overview of the history, approaches, and related works are discussed below.

Classical Approaches

Style transfer has been a field in development ever since the pioneering work published by Gatys et. al in 2016 [5]. The original traditional approaches were often optimization-based methods, which used pretrained feature networks for feature extraction. For example, many early papers utilized the VGG-19 network as their feature extractors, which was very popular in the mid-2010s [11, 16]. Some of these papers built off the concept of calculating the Gram matrix (from the original Gatys paper) in order to calculate a global overview of features in an image, which does not take into account the spatial understanding of features within the image. Other techniques aimed to take a different approach to solving this problem, by aiming to match image styles based on localized image patches, which brings in the added benefit of spatial understanding to sections of the images [1, 16].

Soon thereafter, researchers began to explore learning-based approaches to photorealistic style transfer, compared to the traditional optimization-based approaches above. These approaches introduced the concepts of experimenting with different loss functions, pre-processing, and post-processing the images, and overall focused more on the data-driven aspect of the style transfer [13, 15, 25].

Advancements on Other Fronts

Regardless of which approach was taken, these techniques were all cutting edge at the time of their publication, and various future papers aimed to build off these works. Going beyond merely the exact approach taken, however, two other key factors were always of importance for improving these style transfer image generations. One group of techniques aimed to speed up the models by increasing their computational efficiency and operations, since they recognize speed is often a limiting factor that is valuable when it comes to the applications of these models [8, 12]. Other approaches instead built off the existing baselines to focus on improving the quality of image generations, even if it came at the expense of speed in some situations [14, 22, 24]. Hence, speed and quality have always been working in parallel as two key considerations for all innovations in this field, and striking an optimal balance between the two is the ideal outcome that researchers seek.

Transformer-Based Models

Eventually, as the field of natural language processing evolved, an older paper introducing the ”transformer” architecture began to quickly rise in popularity and gain prominence as a major breakthrough in the field [21]. Naturally, the model quickly found its way into applications for computer vision, and the ”vision-transformer (ViT)” was born [4]. Some efforts were soon made into applying ViTs towards style transfer. One of the key capabilities of transformers is the ability to capture long-range dependencies within data, so some researchers leveraged this technique to reformulate style-transfer across sequential patches of images, using ViTs to convert these images taking advantage of this inherently linked sequential nature [3]. Other approaches simply utilized a pretrained and fixed ViT model to use as an external semantic prior, thereafter using these features in order to splice together the content and style representations together [19].

Diffusion-Based Models

Most recently, diffusion models have been taking the Generative AI world by storm. They were first introduced in 2015 at Stanford and significantly improved upon in 2020 at UC Berkeley, serving as the backbone behind generative models that are ubiquitous across academia and industry today [18, 6]. Recent works relevant to our project aim to apply diffusion-based techniques in order to perform photorealistic style transfer. Some approaches involve aiming to learn the artistic style from just one image, and thereafter using that in order to automatically generate a learnable textual description for performing style transfer which could be prompted for future input images [26]. Other techniques forego the textual input description entirely, and simply use one single content and style image in order to directly infer the style information between the images without any text-based prompts [23].

Overview and Summary

Overall, there have been a plethora of techniques that can be applied to accomplish the goal of photorealistic style transfer, ranging from the classical traditional techniques early on, to the most cutting-edge heavy methods in recent years. Ultimately, however, the optimal approach we should take depends entirely on the specific use-case and problem we aim to solve given our constraints. Hence, we proceed forth to the next section in order to discuss these key design considerations for our project.

Chapter 4

Design Considerations

The first step to approaching the problem we are trying to solve is to understand the constraints of our current design and setup.

4.1 Data Limitations

A common paradigm in machine learning is that the better your dataset inherently is, the better your model will be able to learn and perform. However, in the current tech stack for AI Racing Tech, there have only been very few datasets collected in the real world to begin with, as the team and competition are still relatively young. As such, we face two types of issues that we need to address, namely the Dataset Size and the Dataset Diversity.

Dataset Size

In total, the current size of the dataset containing real-world images taken on the race car driving through any race track stands in the order of a few thousand images. Given this scarcity of data, the team was thus far required to train the object detection models on all the available data, and unable to meaningfully hold out any data for a proper test set due to limited size. Although other teams were willing to share their image datasets with one another, due to differences in camera calibration across every team's vehicle, the images often appeared visually distinct and were unable to translate to our specific race car's cameras.

Dataset Diversity

Furthermore, even among the already limited real-world racetrack data, there were only a few hundred images (correlating to around one to two minutes, total) which contained a singular other vehicle in view. This means that the rest of the dataset entirely contains only our vehicle driving on the track, and no other vehicles racing alongside or ahead in view. The number of unique race tracks is also limited, as the data was only collected across one or two unique tracks. As a result of all these factors, the dataset diversity is extremely poor and limited, which further limits the quality of our vision models in their current state.

Impact on Model Performance

Considering the above factors, the team is significantly bottle-necked by the dataset size and diversity. Nonetheless, the high expectation of the vehicle is to prepare to race on new domains and race tracks, such as the famous Monza Circuit located north of Milan, Italy. In addition, given the current datasets available, we are unable to provide the car with the ability to learn what it will be like to race with other vehicles concurrently on the track. At best, we are able to include a few frames of our single-competitor data, but the expectation of the vehicle is to race similar to other competitions, oftentimes with two or more other race cars racing in parallel at high speeds.

Beyond the impacts of our limited dataset diversity, the most pressing concern is the dataset size in total, which makes training models to provide high accuracy significantly difficult. Transfer learning based approaches can be used to mitigate this issue, but given the brand new domain of autonomous vehicle racing, nothing will truly come close to having real-world multi-car racing data that we can train and optimize for in-house.

As such, only one clear answer stands out: using simulation data to augment the dataset. We will unlock the ability to create infinitely many scenarios as we desire, simulating multiple vehicles on the race track, and simulating other racetracks as well. Simulation data will be able to solve both of our limitations above, and ultimately yield significantly more data outnumbering the number of real-world examples. Thus, in order to perform the best in a subsequent real-world environment, this makes it all the more paramount that the sim2real approach implemented needs to perform as the highest photorealistic level, lying along the same visual domain as the real-world data for our vehicle's specific camera configuration and appearance. In the short-term/near future, this approach will be perfect. However, not always will this be the case, as we must be versatile to change in the likely event that our cameras are tuned differently over time into the future. This brings up our second key design consideration: adaptability to change.

4.2 Adaptability to Change

The current camera configuration and tuning of various parameters has been decided upon by the team for the time being, and set in stone for the upcoming 1-2 years. However, thinking longer term, the possibility is clear that the cameras may be upgraded, tuned differently, and as a result the images will visually appear to be different. When this change inevitably happens, all our prior existing real-world data and models will become instantly obsolete, as they appear visually and inherently different than the new images from camera hardware changes.

As a result, this is another consideration that must be taken into account when deciding the sim2real approach we take, in order to convert all our simulation data to the new real-world domain to train our models.

4.3 High-Level Model Approaches

From a high-level, there are two key ways to tackle this problem. The first involves intensively pretrained models, while the second involves test-time training models, in order to perform the Sim2Real generation for our data.

Pretraining-Based Models

Pretraining-based models often involve feeding a model with plentiful examples of images, and tuning the model to perform well on the data it is given. The benefit of this type of model is that the majority of the time is spent in the pretraining phase, which means that the test-time output generation is often very quick and efficient. It may take hours to train the model, but once it is finished training, converting simulation images to sim2real images can occur rapidly and efficiently.

However, the requirements for such types of models are that there often needs to be an extensive amount of data in the training process, in order for the model to learn what the simulation and real domains look like. While simulation data is plentiful in our case, the major issue lies in the real-data domain, where the exact opposite situation is present. As such, it will be very difficult to train an intensively pre-trained model, simply due to the fact that there may not be enough real-world image data for the model to learn from in the first place. In addition, using data online from ImageNet or publicly available datasets will not help, since we specifically want the outputs to be as close as possible to the visual appearance of our specific camera hardware.

Test-Time Training Models

On the other hand, another valuable approach lies in the concept of test-time training models. The benefit of this approach is that there is zero effort needed in order to create a model for a specific style, but rather it is learned directly at test-time from the style image provided. Another added benefit of this approach is that, oftentimes, only one single baseline style image may be needed in order to perform style transfer on infinite numbers of images, which is extremely valuable in data-constrained environments (such as our own) where image data may not be plentiful and is open to change.

However, the drawback of these methods is that rather than taking time for intensive pretraining, each image needs to be generated and refined over iterations, meaning the time cost comes on the tail end of the style transfer process. Furthermore, the style transfer output can often depend heavily on the single style image provided, meaning that there may need more empirical testing in order to find proper images for creating optimal outputs.

4.4 Proposed Architecture Model for Sim2Real

After considering all the factors presented above, especially in regards to data limitations and the necessity of adaptability to change, the best solution clearly lies within the test-time training models. The reason for this decision is because we have limited training data of real-world images, which may make certain pretraining-based models difficult to create and train. Furthermore, even though more data is actively being collected over time, our camera hardware team will likely make changes and tune camera parameters as the technology, funding, and project evolves, meaning that new visual image data on future camera configurations will essentially require us to start from scratch, and completely render any prior sim2real models obsolete.

Diving deeper within the branch of test-time training models, there are also various factors to consider. In particular, one of the most important considerations is the time taken to generate images, and perform the photo-realistic sim2real style transfer. As a result of this, we aim to find a model which can perform the fastest sim2real conversion, even if it may be at the expense of a little image output quality. Looking at the related works, both diffusion models and vision transformers produce the most impressive visual outputs in regards to quality, but through empirical testing we noticed that many of these models often require significant compute resources, or conversely significant amounts of time on limited compute. Due to lab hardware constraints, we fall into the latter category, meaning that both of these approaches are not suitable at this current moment of time for our team.

Through continued research, one very interesting paper came up from 2022, titled the "Deep Translation Prior: Test-time Training for Photorealistic Style Transfer" [9]. Upon further inspection of this paper, we realized that it perfectly addresses all the constraints of our project. As the title implies, the model involves a test-time style transfer model. Furthermore, through empirical testing we were able to determine that the model can run very efficiently on a single GPU in our laboratory, within a mere minute or two. In addition, due to the similarity of our sim and real images both involving race cars and race tracks (unlike the vastly different content-style realms covered by the paper), we recognize that we can likely reduce the number of iterations required to perform our sim2real transfer, further reducing the time cost in order to generate our outputs. Finally, a publicly available codebase allows for us to rapidly jump into implementation, modifying and customizing the model architecture for our specific use-case in order to quickly generate, iterate, fine-tune, and streamline our data generation process. Hence, our project will proceed forth with the Deep Translation Prior model architecture.

Chapter 5

Project Implementation

5.1 Data Collection

The first step of our project implementation process is data collection, which in our case begins with the simulation engine. The exact steps that were taken are outlined below.

Simulation Engine

For simulation, we are using the OSSDC (Open Source Self Driving Car) Simulator, which is a self driving car simulator based of the now-deprecated LGSVL (LG Silicon Valley Lab) Simulator. A lot of engineering work went into setting up the simulator, the computer hardware GPUs to run the software, and generating the data. Over the course of a few weeks, we were able to successfully simulate multiple vehicles in the simulator environment. The sensor setup (in particular, the onboard cameras), were designed to be a digital twin of our real-world vehicle. As such, we then logged various simulated race car runs in the simulator, extracting data from the ”Vimba Front Left Center” camera onboard the vehicle. To create edge case scenarios and expand our data, addressing the dataset diversity issue we were facing, we automated the driving process for the chase vehicle, and took over manual control for the two vehicles in view. Commands were sent to these two other cars through the terminal, running a separate Docker instance for every single individual agent in the scene. This allowed us to have limited manual control over the other vehicles' speed, trajectory, and paths that they would follow. Using a clever combination of commands perfectly timed throughout the track recording duration, we were able to capture complex movements such as cars interweaving paths with one another, vehicles overtaking and cutting each other off, vehicles coming into extremely close proximity to the chase car, the chase car overtaking other vehicles from both the left and right angles, and even a collision case scenario in which one vehicle pit maneuvers the other vehicle off the track. Select images from these instances we generated are visualized below.

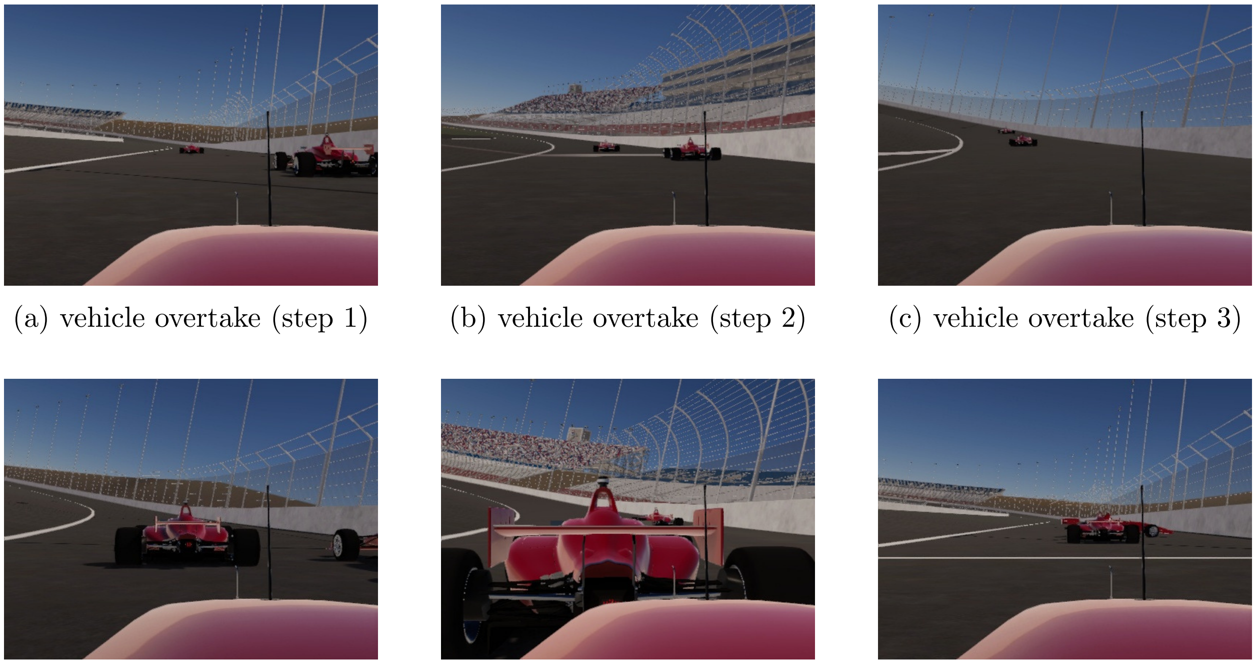

(a) vehicle overtake (step 1)
(b) vehicle overtake (step 2)
(c) vehicle overtake (step 3)
(d) vehicle weaving between
(e) extremely close proximity
(f) accident collision scenario

Figure 5.1: Sample images of diverse multi-car scenarios for our dataset

In addition to collecting data with various configurations and movements of the vehicle, we also sought to generate data in unseen domains. Thus, progress has been made and is further underway for recreating the Monza Circuit in Italy, being able to create race track boundaries and paths, and simulating multiple vehicles in this more complex environment. Shown below are preliminary images on the simulated Monza race track, through which we will continue to collect more data in the future.

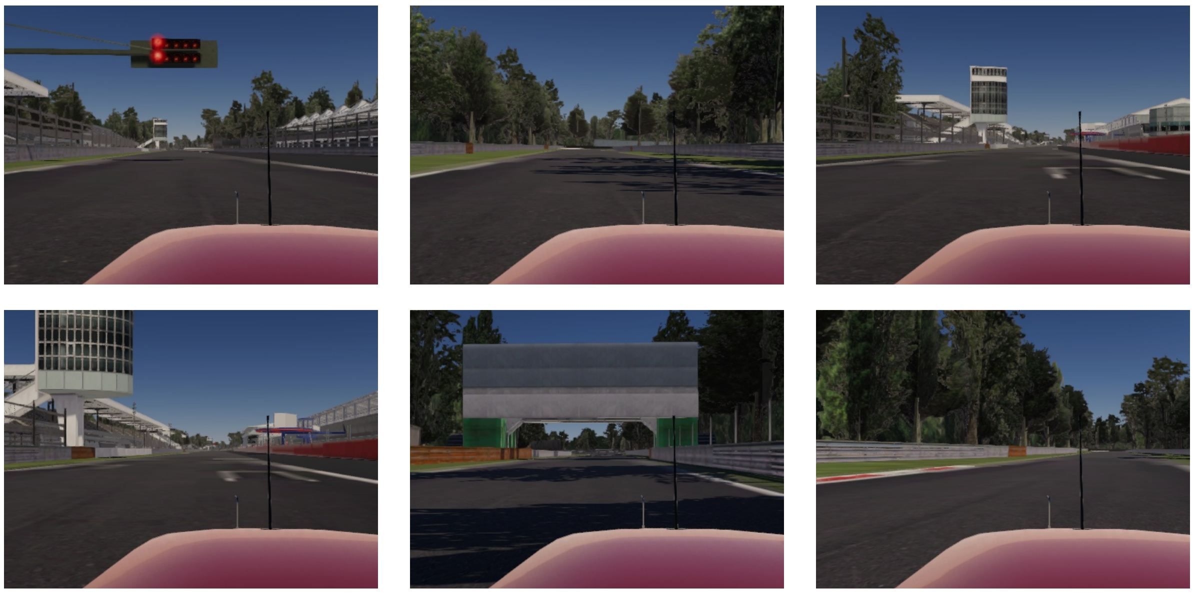


Figure 5.2: Sample images from the simulated Monza Circuit racetrack

5.2 Sim2Real Model

Model Architecture

The Deep Translation Prior (DTP) model architecture can be broken up into two key components, the correspondance module and generation module, shown in the architecture diagram below:

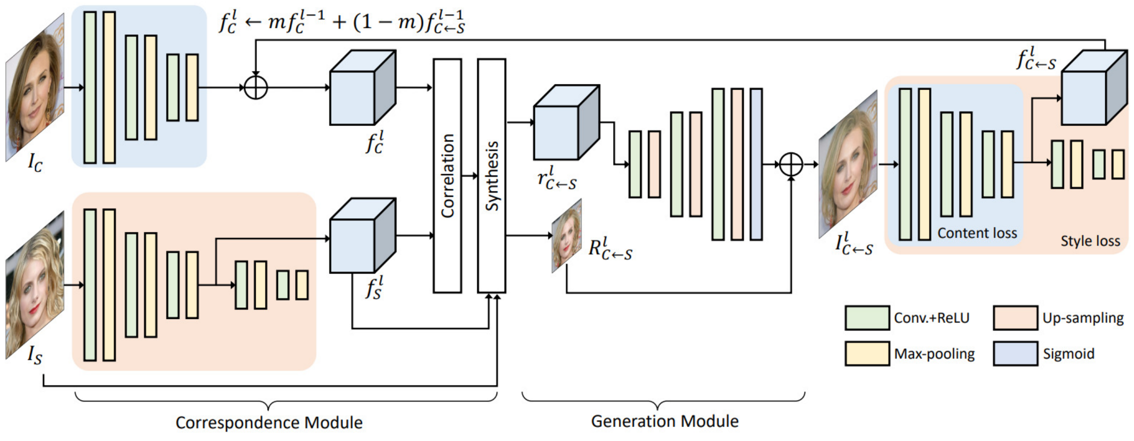


Figure 5.3: Deep Translation Prior (DTP) Model Architecture Diagram

This model essentially works by taking in one single input image for the content (a simulation image), and one single image for the style (a real world image). Using these two images alone, the model goes through various iterations in order to iterate and generate the optimal output.

At its core, the network architecture consists of two sub-modules, namely the correspondence module and generation module. The correspondence module essentially serves to predict a translation hypothesis by calculating the similarities between source and target points in the two images, and create a warped feature image off the style input. Thereafter, the generation module takes this warped feature input and passes it through a decoding network, in order to generate the final residual of the converted stylized image.

The Deep Translation Prior paper utilizes a clever combination of three loss functions: a content loss $\mathcal{L}_{cont}$, a style loss $\mathcal{L}_{style}$, and a cycle consistency loss $\mathcal{L}_{cyc}$. Thereafter, each of these losses are weighted in order to compute the total final loss, defined by the following formula: $\mathcal{L} = \lambda_c \mathcal{L}_{cont} + (1 - \lambda_c)\mathcal{L}_{style} + \lambda_{cyc}\mathcal{L}_{cyc}$ (where λ_c and λ_{cyc} are adjustable parameters).

Using the approach above, alongside a plethora of other parameters that can be modified in the model architecture (which will be explained and explored further through the experimentation chapter below), we can aim to achieve optimal outputs tailored for our task.

5.3 Image Labelling

In order to label the images, the easiest and most standard way to go about the process is to draw bounding boxes around all race car vehicles visible in every frame. Although this is a tedious task, it is unfortunately one that must be done, and as a result required manual hand-labelling of every single individual frame across all rosbags.

In order to speed up the process, we use an open-source software called Computer Vision Annotation Tool (CVAT), which streamlines the process of loading up, labelling, and exporting image-label pairs thereafter [2]. Some benefits of CVAT is that given bounding box labels for two frames that are 10 timesteps apart, it is able to try its best in order to interpolate between frames, and label intermediate frames as well. In practice, this wasn't always perfect and often required additional adjustment by hand, but ultimately provided flexibility and ease of use in bounding box labelling.

5.4 Post-Processing

Once the images are labelled, we now have the outputs stored as an image-label pair of files containing corresponding bounding boxes for each of the objects (race cars) visible in the scene. These bounding boxes provide a precise location of the car in the scene.

Converting Bounding Boxes to Segmentation Masks

However, the team's requirement is to create a model capable of performing instance segmentation on the vehicles, providing a tighter and more fine-grained outline of the car on the track (compared to a general bounding box). As a result, we must convert from bounding boxes to segmentation masks, which is a traditionally difficult task. Common classical techniques in the past have involved everything from painstakingly hand-labelling every single individual pixel by hand, to edge detection models that would infill between detected boundaries. However, we live in the 21st century, and deep learning models are the boon of our existence. Meta (formerly known as Facebook) released their groundbreaking "Segment Anything Model (SAM)" which has made image segmentation significantly more efficient.

Segment Anything Model (SAM) by Meta AI

The Segment Anything Model was created by Meta's AI Research team, and was trained on 11 million images and 1.1 billion segmentation masks [10]. From a high-level overview, the model is composed of two components. The first is a featurization transformer module that compresses an image into a rich 256x64x64 feature matrix. Thereafter, this matrix is passed into a decoder head, which further takes in the model's prompts (a general outlined mask, specifically labelled points, or a text input), and combines them together to output predicted instance segmentation masks. The architecture can be seen in the figure below:

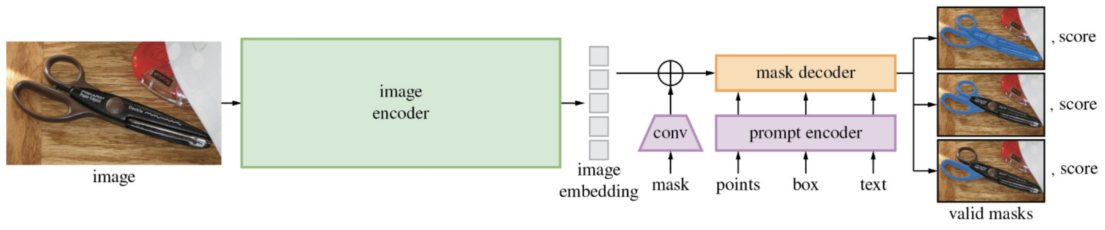
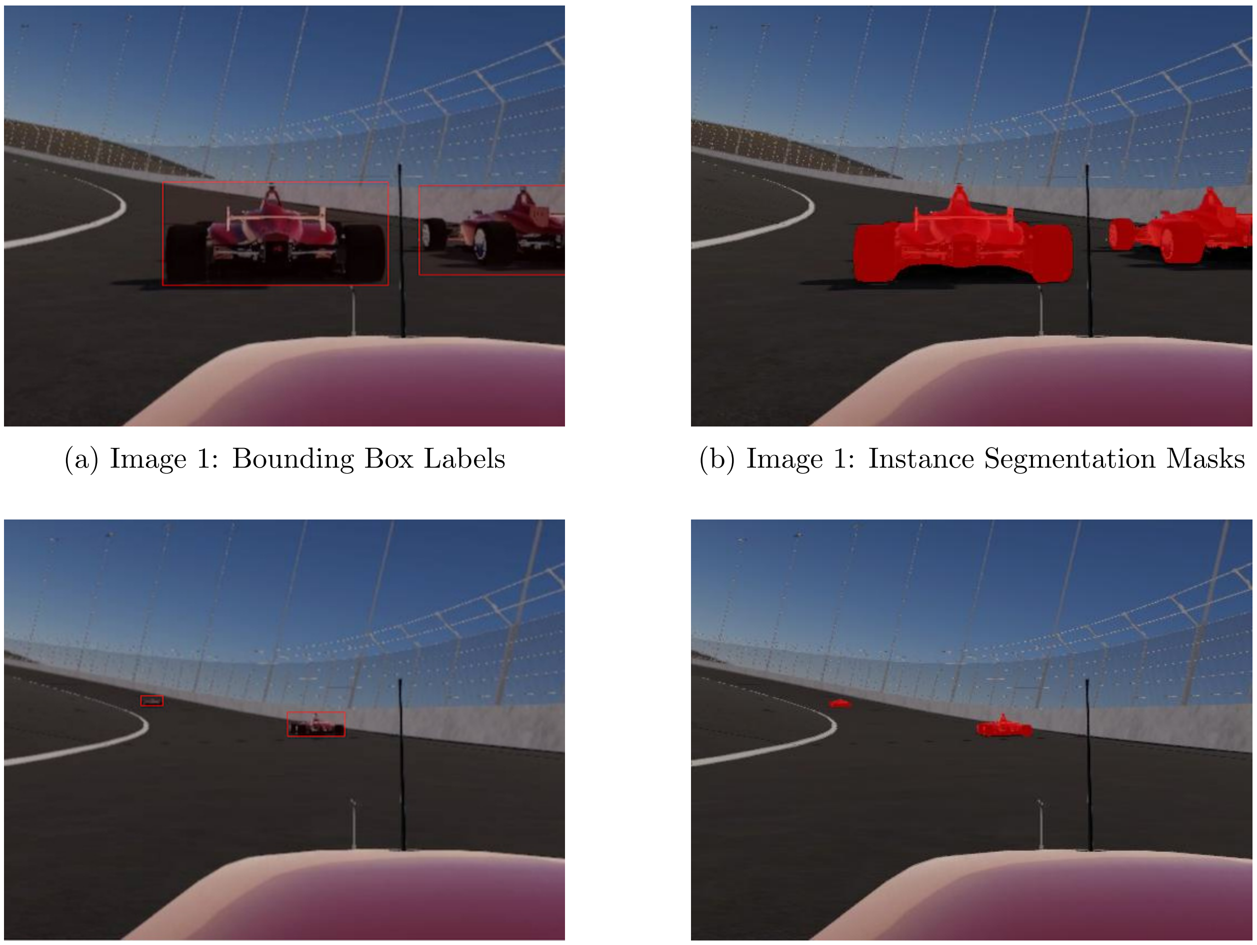


Figure 5.4: Meta's Segment Anything Model (SAM) Architecture Diagram

Coming back to our use-case, we leverage SAM's capabilities in order to convert our bounding boxes into segmentation masks. The approach we took is to input the simulation image and corresponding bounding box labels into the model, and then auto-select the center-point of each bounding box as a keypoint for SAM, since it represents the exact middle of the race car body in view. Thereafter, SAM automatically segments the image, and generates instance segmentation masks for each vehicle in the frame. Later on in the verification process, individual frames are manually parsed by hand to double-check the results, and in a few cases the segmentation mask will undergo further modifications. This is done by repeatedly adding more points for areas the model missed, or setting removal points for extraneous regions that were erroneously added to the segmentation. We have visualized some of these results below:


(a) Image 1: Bounding Box Labels

(b) Image 1: Instance Segmentation Masks

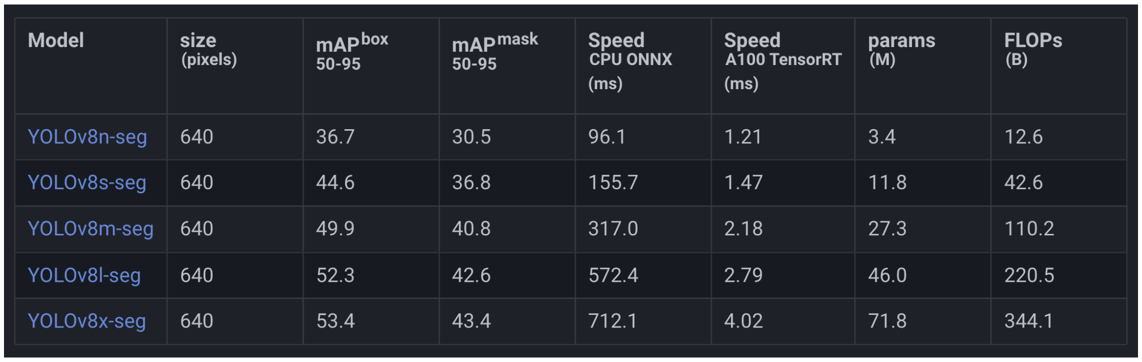
(c) Image 2: Bounding Box Labels

(d) Image 2: Instance Segmentation Masks

Figure 5.5: Converting Bounding Boxes (Hand-Labelled) to Segmentation Masks (SAM)

5.5 Segmentation Model Training

The final stage in the pipeline is to train the object detection and instance segmentation model itself. In our case, due to the fast-paced nature of race car driving, we opted for one of the best real-time models, YOLOv8. YOLOv8 is the 8th version of the "You Only Look Once" Model first published in 2015, and the model (along with pretrained weights) is publicly released for use by the software company Ultralytics [17, 20]. The core new capability that it introduced over its predecessor YOLOv7 is support for segmentation, breaking beyond the standard bounding box detection it has been limited to in the past. As a result, this model was a perfect choice for our project. To accomplish our task, we implemented a transfer-learning based approach, utilizing the pretrained models and weights available on the website.



Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n-seg	640	36.7	30.5	96.1	1.21	3.4	12.6
YOLOv8s-seg	640	44.6	36.8	155.7	1.47	11.8	42.6
YOLOv8m-seg	640	49.9	40.8	317.0	2.18	27.3	110.2
YOLOv8l-seg	640	52.3	42.6	572.4	2.79	46.0	220.5
YOLOv8x-seg	640	53.4	43.4	712.1	4.02	71.8	344.1

Figure 5.6: YOLOv8 Publicly Available Models and Weights

Looking through the options available above, we needed to decide which model would be best to proceed forth for our vehicle. There are various factors to consider, in particular being the clear trade-offs between inference speed, model size, and accuracy. In the case of autonomous race cars, vehicles can reach driving speeds of up to 200 miles per hour, which means that there will often be split-second decisions that mean the difference between staying alive on the track vs. colliding with other vehicles (a very, very expensive mistake). Hence, we choose to opt for the most lightweight model available, YOLOv8n-seg, due to its extremely quick inference time with relatively good accuracy for our use case.

Chapter 6

Experimentation

Now that the model architecture has been finalized to the Deep Translation Prior model, there are a variety of experiments we need to run. Inherently, the model can be tuned based on different parameters that are passed in as arguments to the training pipeline. Upon further inspection through the code, the key parameters that would have the greatest visual impact on the output are:

- `style_root`: the real (style) images to use as our baseline
- `iter`: the number of iterations for optimization
- `content_style_wt`: the weighting between content and style
- `cycle_wt`: the weight of cycle consistency regularization

The goal is to ultimately converge upon the best set of parameters, images, and tuning for our specific use-case that will maximize the realism of our generated sim2real outputs, and subsequently yield the highest accuracy instance segmentation models that will be deployed onto the race car. In an ideal world, the optimal process is a purely grid search approach, trying every combination of parameters for the factors above and training the final models to see which outputs the highest accuracy on average. However, this approach is far too infeasible in terms of the time and compute required to generate images and training new models for every combination. Instead, a better approach would be to first use a set of human visual inspections, checking how the outputs for each set of parameters appear. Typically, it should be pretty quick to recognize when a certain set of parameters produces poor results that look unrealistic, and we can save time by filtering out these parameters from future experiments. As we encroach upon the final few combinations, in the chance that the sim2real output images become visually indistinguishable to the human eye, we will then resort to training individual models, and quantitatively compare the output accuracy across the final selection.

6.1 Baseline Real Style Image: `style_root`

We first test the effects of using different baseline real (style) images on the output sim2real generated results. For our experiment, we hand select various random frames throughout a real rosbag of our vehicle's camera. We test on the standard model parameters, to perform an unbiased comparison between the sim2real outputs across different baseline images. The results are visualized below:

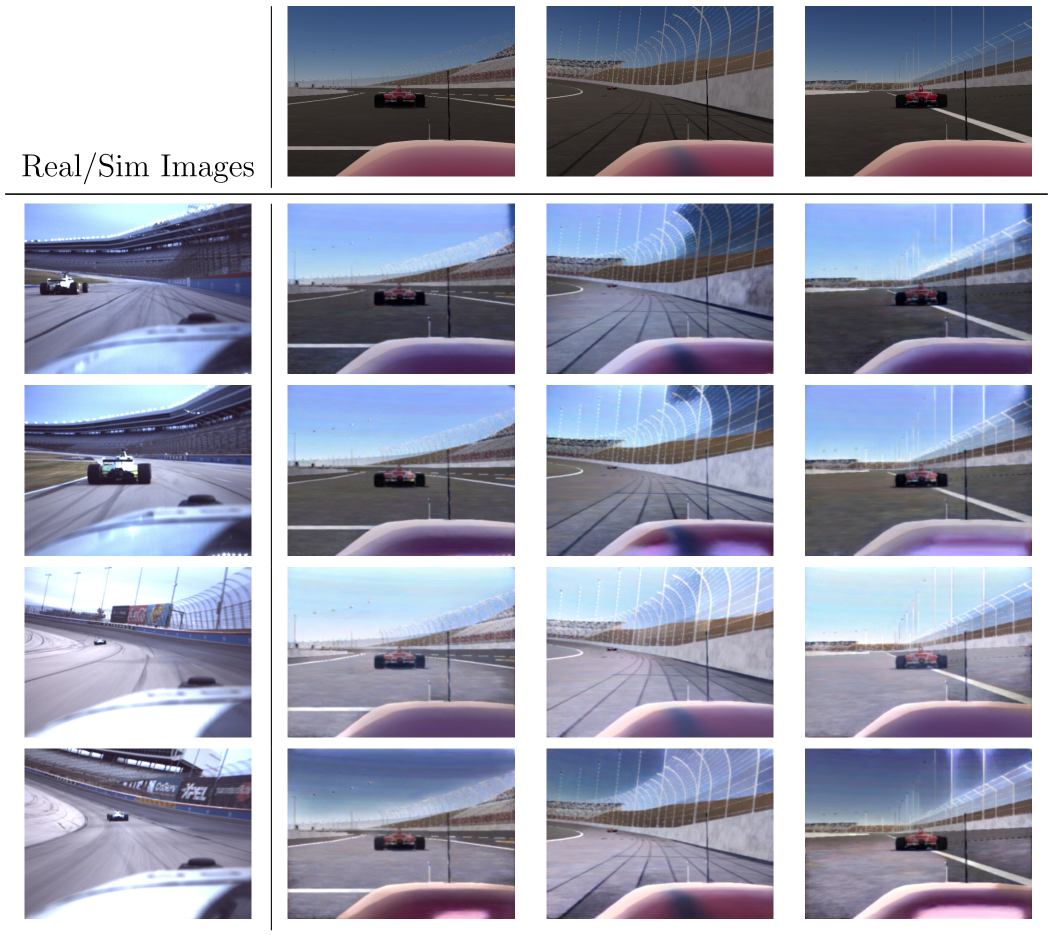


Figure 6.1: Experimenting with Real Images (Rows) and Sim Images (Columns)

Starting with the 1st real image above, we can easily see very clear artifacts in the 2nd sim image across dark patches on the fence. The 2nd real image contains the same issue as well. The 4th real image causes unnaturally dark skies which look unrealistic. Ultimately, the 3rd image produces the best outputs, with visually consistent track color, sky color, minimal fence artifacts, and overall matching best to the real image style. Hence, we will proceed forth with the 3rd real image as our baseline style image for future sim2real generations.

6.2 Content-Style Weight: `content_style_wt`

We next test the effects of modifying the `content_style_wt` on the output sim2real results. For our experiment, we iterate over values of 0.2, 0.4, 0.6, 0.8 (default), and 1.0. The reason for testing more lower rather than higher values (compared to default) is because we want the outputs to bias towards the style (real) image rather than content (simulation), in order to look as visually close as possible to our onboard camera. The results are visualized below:

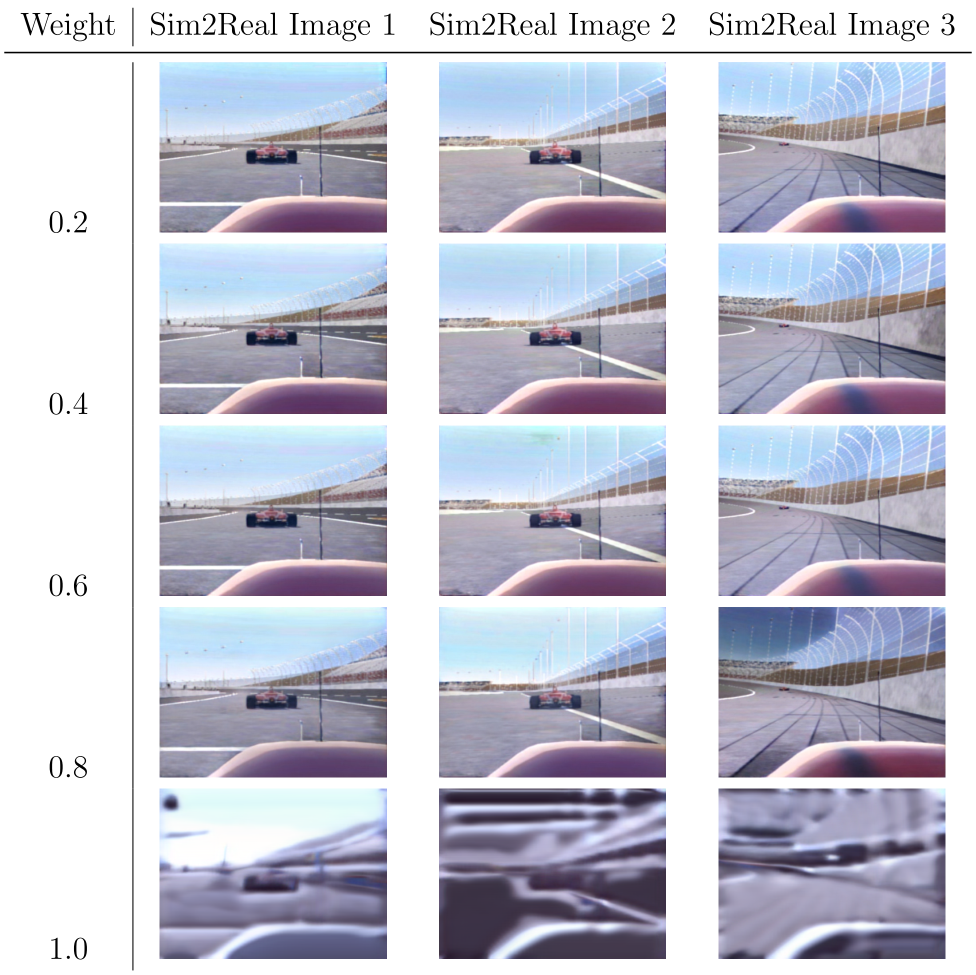


Figure 6.2: Experimenting with Content-Style Weight Value

Right away, we can see that 1.0 produces extremely poor outputs, meaning we can discard it. Similarly, 0.8 produces extremely unrealistic skies for certain images, either too bright or too dark. The 0.6 value suffers as well, with some green artifacts on the middle image in the sky. Both 0.4 and 0.2 look impressive. However, in further rigorous visual testing (not pictured above), we found 0.4 to provide the most consistent results, with minimal visual artifacts in diverse conditions. Hence, we will proceed with 0.4 for our `content_style_wt`.

6.3 Number of Iterations: `iter`

Moving on, we now test the effects of number of iterations on the output sim2real generated results. For our experiment, we iterate over iteration counts of 200, 400, 600, 800, and 1000 (default). The reason for testing the lower end of iterations is because outputs are generated faster with lower iterations, which is a key factor to consider as previously mentioned.

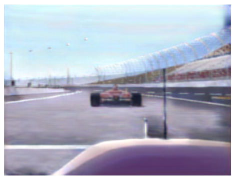
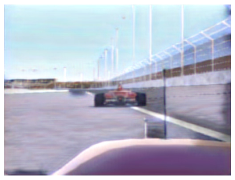
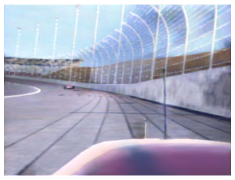
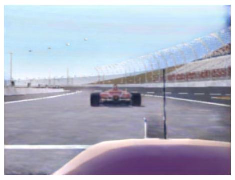
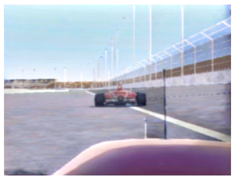
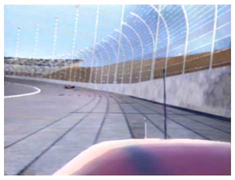
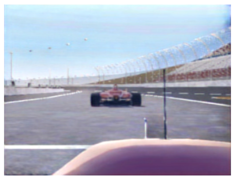
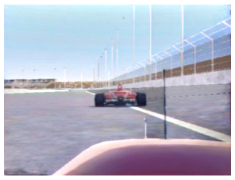
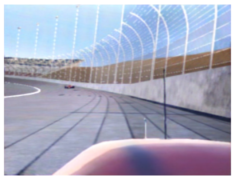
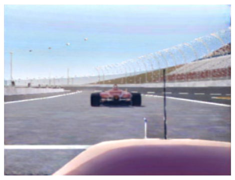
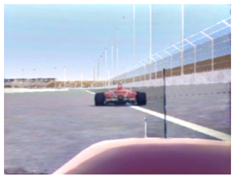
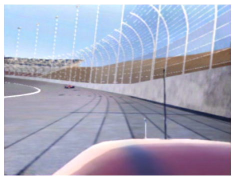
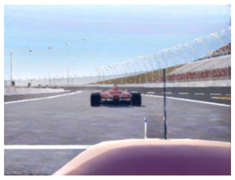
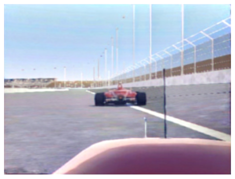
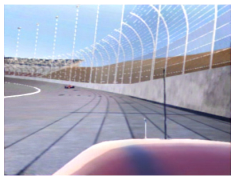
Iterations	Sim2Real Image 1	Sim2Real Image 2	Sim2Real Image 3
200			
400			
600			
800			
1000			

Figure 6.3: Experimenting with Number of Iterations

We can instantly see that 200 iterations is far too little to generate clean outputs, which was to be expected. 400 similarly suffers from poor generation, especially on the 3rd simulation image. 600 is slightly better, but on the 1st simulation image we can see artifacts in the background landscape. However, looking between 800 and 1000 iterations, the outputs seem virtually indistinguishable, which was confirmed through further visual testing as well. Hence, 800 will save significant time at scale for image generation compared to 1000. Thus, we will proceed forth with 800 iterations.

6.4 Cycle Consistency Regularization: `cycle_wt`

For our final set of experiments below, we test the effects of modifying the `cycle_wt` on the output sim2real results. For our experiment, we iterate over values of 0.0 (no cycle consistency), 0.5, 1.0 (default), and 1.5. We aim to test an even spread of values, in order to see whether less or more regularization weight penalty will lead to better sim2real outputs.

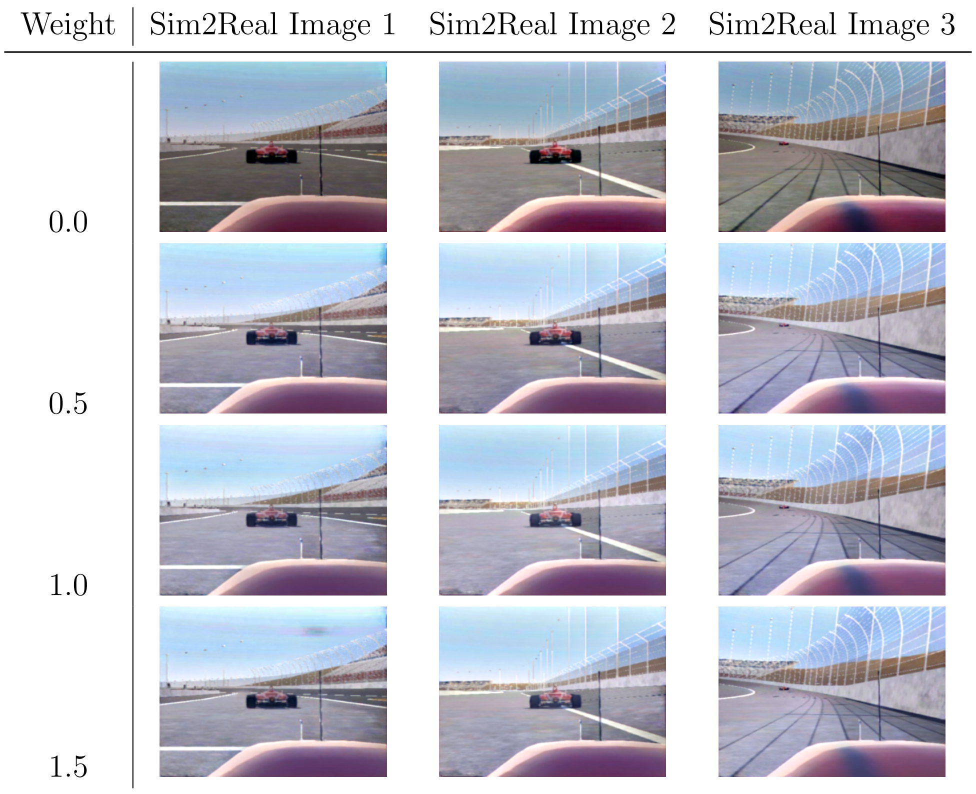


Figure 6.4: Experimenting with Cycle Consistency Regularization Weight Value

Looking at the results, we can instantly recognize that a weight of 0.0 (effectively removing cycle consistency) creates unrealistic sim2real outputs without learning from the style (real) image. On the other hand, the higher end of 1.5 produces visual artifacts in the 1st image's sky and 3rd image's fence. The 0.5 value initially seems visually indistinguishable from 1.0. However, upon closer inspection we see that 0.5 produces visual artifacts at the edges of the image, which was confirmed across other visual generations as well. Hence, ultimately it seems that the default value of 1.0 for `cycle_wt` provides the most consistent results, with minimal visual artifacts in the output across diverse sets of lighting conditions, race track textures, and more.

After completion of the above experiments, we have now finalized our parameters, and will proceed forth with these values in our image generation and model training ahead.

Chapter 7

Results

The final step to evaluating the outcome of our approach is to objectively assess the results, and see whether we receive performance gains from utilizing sim2real data.

7.1 Dataset Structure

We will be using various datasets in order to train different models, and the information regarding each group of data is described in detail below.

Datasets Breakdown:

- **Dataset A:** For our simulation image training/validation dataset, we have collected two rosbags of three race vehicles (two excluding the chase car) running various laps around the track, adding up to a total of 1184 images.
- **Dataset B:** For our sim2real image training/validation dataset, we take every image from Dataset A, and convert them through our sim2real model pipeline using the best parameters converged upon in the previous section. As a result of this approach, every single simulation image in Dataset A has a 1:1 pair in the sim2real Dataset B, with identical layouts, car positions, etc. Thus, this dataset also has a matching total of exactly 1184 images.
- **Dataset C:** For our real image training/validation dataset, we have collected two rosbags of two race vehicles (one excluding the chase car) through various laps, adding up to a total of 703 images.
- **Test Dataset:** For our real image test dataset, we have two rosbags of two race vehicles (one excluding the chase car), adding up to a total of 1434 images. This test dataset is entirely a holdout set, from a completely different day and track run than the real images used in training/validation. This ensures there is no internal bias in the model evaluation, and all models are fairly evaluated on an identical unseen domain.

7.2 Evaluation Setup

As stated earlier in the paper, we will be utilizing YOLOv8 as our model of choice to perform instance segmentation on the race track. In order to create the most objective comparison of data improvements, a total of five models must be trained and evaluated:

YOLOv8 Models to Train:

1. YOLOv8 trained on real data ONLY **(Dataset C)**
2. YOLOv8 trained on simulation data ONLY **(Dataset A)**
3. YOLOv8 trained on simulation + real data **(Dataset A + Dataset C)**
4. YOLOv8 trained on sim2real data ONLY **(Dataset B)**
5. YOLOv8 trained on sim2real + real data **(Dataset B + Dataset C)**

Furthermore, in order to make the model training and evaluation as objective as possible, we will define all other variables, parameters, and factors to be completely identical between the experiments. As a result, all YOLOv8 models will be trained using the following set of parameters:

YOLOv8 Training Parameters:

- **Model Architecture:** YOLOv8 (from Ultralytics)
- **Pre-trained Weights:** yolov8n-seg.pt (from Ultralytics)
- **Number of Epochs:** 25
- **Image Size:** 512 x 384
- **Batch Size:** 16
- **Load Best Weights:** True

All other parameters not mentioned above, such as the learning rate, momentum, optimizer, etc. are kept as the same standard default values across all models.

7.3 The Ideal Outcome to Prove Sim2Real Success

In theory, the ideal expected outcome to prove the benefits of sim2real would entail that the test detection and segmentation accuracy vary in specific ways when comparing a few models at a time. First and foremost, we should expect Model #2 to have the least accuracy, since it is trained purely on simulation data and thus should have the least visually similar output to the test data distribution, causing it to generalize the least. On the other end of the spectrum, we should expect Model #5 to have the highest accuracy, since combining sim2real data with the original real data allows us to provide every last image we have gathered to the model in order to train, on images that are all realistic, and thus outperform every other model possible that can be trained on our problem.

In specific comparisons, the most objective measure of whether sim2real truly made a difference and improvement over simulation data is if Model #4 (sim2real only) outperforms model #2 (sim only). This is because every image between these two models is a pure 1:1 comparison with the exact same layout, viewpoint, and position of cars, but purely the visual elements of the image have been modified to be more realistic. Hence, if Model #4 beats Model #2, this shows that the sim2real image conversion truly made a difference, and higher test accuracy on real-world data proves that the converted images truly brings realism into the images, bridging the reality gap.

The best measure to assess actionable benefits gained from this work is to compare Model #1 (real-only), Model #3 (sim + real), and Model #5 (sim2real + real). Model #1 represents the state of team's efforts if no simulation engine existed at all, using only the real-world data we have. Model #3 demonstrates the state of the team's efforts before this project was created, leveraging all sources of data we currently have to train our models. Model #5 represents the future potential for our project, converting all new simulation images through the sim2real pipeline, in order to generate and bring realism to all future collected sim images. If sim2real truly is effective, then we would expect the accuracy to increase from left to right, proving to us that every generational step towards improving our models has been a step-up in efficacy as well, and most importantly that the future goals for using sim2real to convert data produces the highest performance we have seen to date.

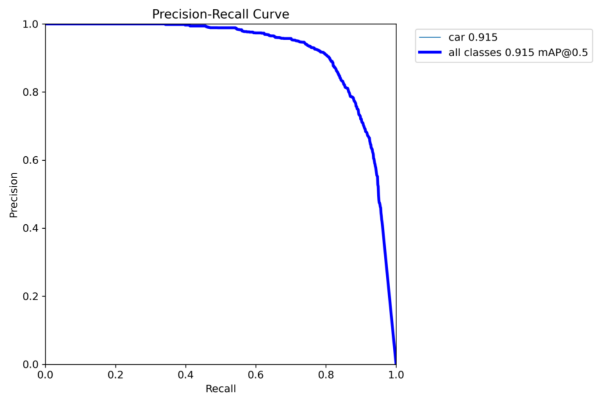
As previously stated, the five models we described were ultimately all trained in the exact same way with identical parameters, and the best final model weights were saved for each one. In evaluating the output, they were assessed both qualitatively as well as quantitatively. A detailed analysis of all these results are discussed below:

7.4 Quantitative (Numerical) Results

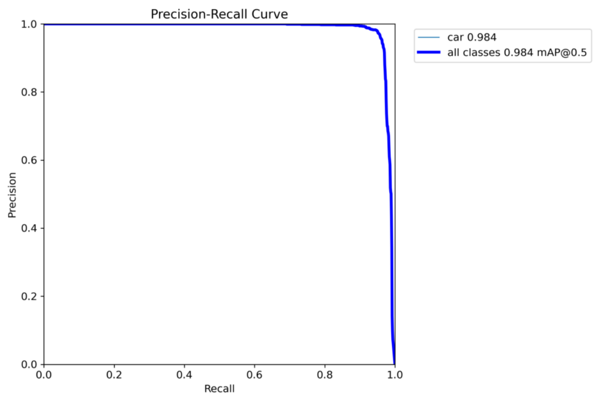
To compare the performance of our models quantitatively, the best standard metric to use in comparison of segmentation models would be the mean average precision (mAP) value. Another key metric is the precision-recall (PR) curve, generated by running the model on the test dataset and comparing generated segmentation masks against the baseline ground truth. A higher mAP value means that a specific model performs better than another.

Model #2 (Pure Sim) vs. Model #4 (Pure Sim2Real)

Comparing Model #2 to Model #4, the precision-recall curves are shown below:



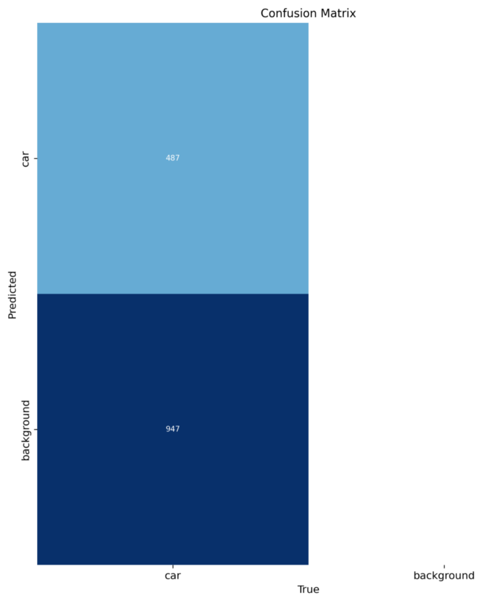
(a) Model #2 Precision-Recall Curve



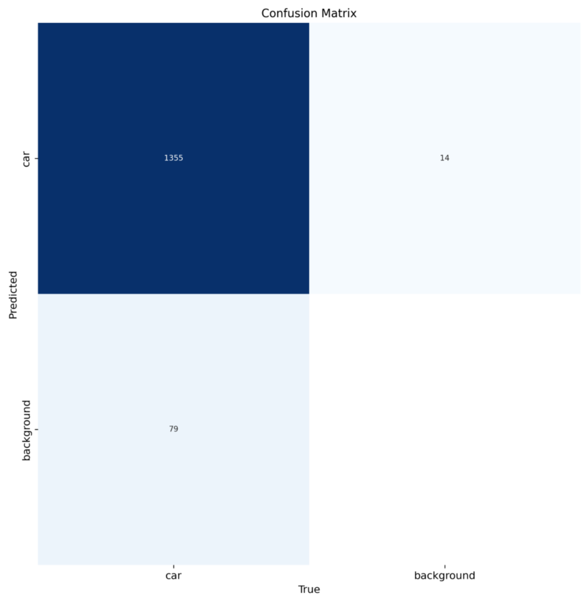
(b) Model #4 Precision-Recall Curve

Looking at the results above, we see that Model #2 has an mAP value of 0.915, whereas Model #4 has an mAP value of 0.984. The graphs also demonstrate that Model #4's PR curve is much closer to the upper-right corner compared to Model #2, meaning that it is capable of achieving a point with both higher precision and recall values at the same time. This significant jump clearly demonstrates that sim2real truly bridges the reality gap better than purely simulation data, as the output test dataset is entirely run on real-world images.

However, simply looking at the PR curves doesn't paint the full picture. In our specific use-case, it is important to know exactly how many frames that the model correctly (or incorrectly) identifies the race car in the scene. Hence, confusion matrices are a perfect way of identifying exactly how many test frames were detected on the race track, shown below:



(a) Model #2 Confusion Matrix



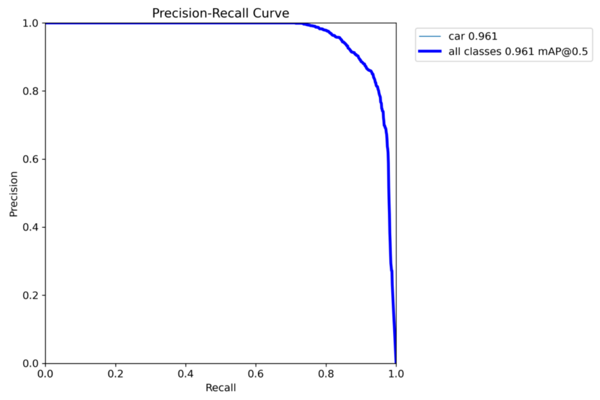
(b) Model #4 Confusion Matrix

Looking at the results above, we can see that Model #2 misses a significant number of frames of the test data, where it correctly identifies the vehicle for 487 frames, and misses it for the remaining 947 frames. It never misidentifies a car on the track, however. On the other hand, Model #4 correctly detects a significantly higher value of 1355 frames of the vehicle, with only 79 frames where it misses the car, and 14 frames where it inadvertently detects a ghost vehicle.

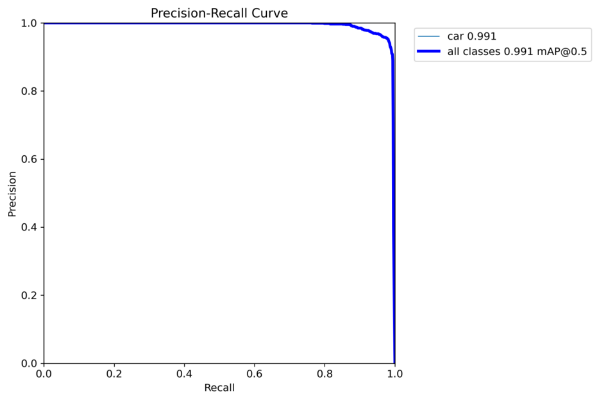
Thus overall, the results above undoubtedly prove to us that sim2real outperforms pure simulation images in their model performance, and that our sim2real model effectively bridges the reality gap!

Model #1 (Pure Real) vs. Model #3 (Sim + Real) vs. Model #5 (Sim2Real + Real)

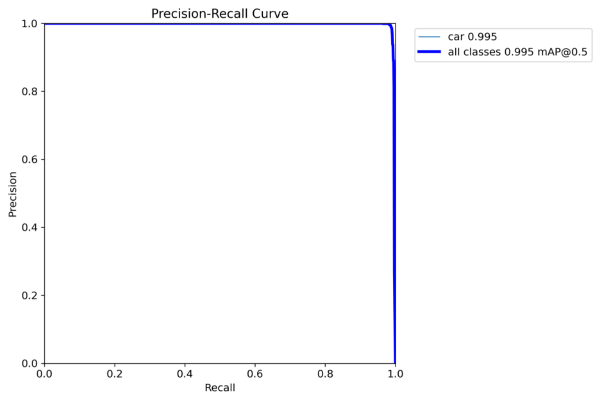
Now comparing Model #1, Model #3, and Model #5, the results demonstrate the following:



(a) Model #1 Precision-Recall Curve



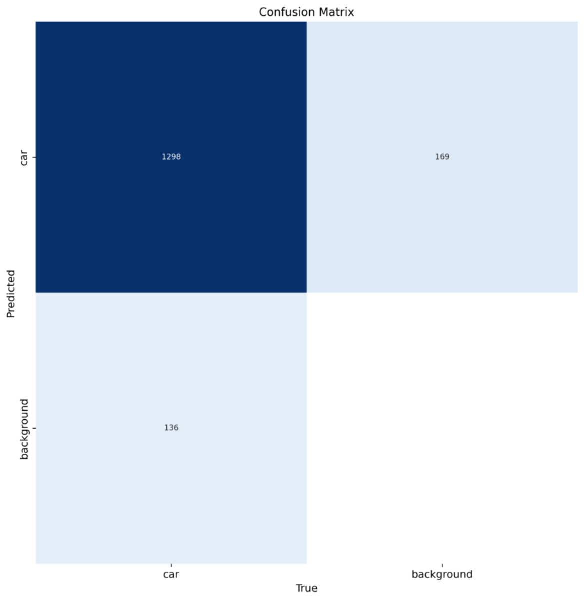
(b) Model #3 Precision-Recall Curve



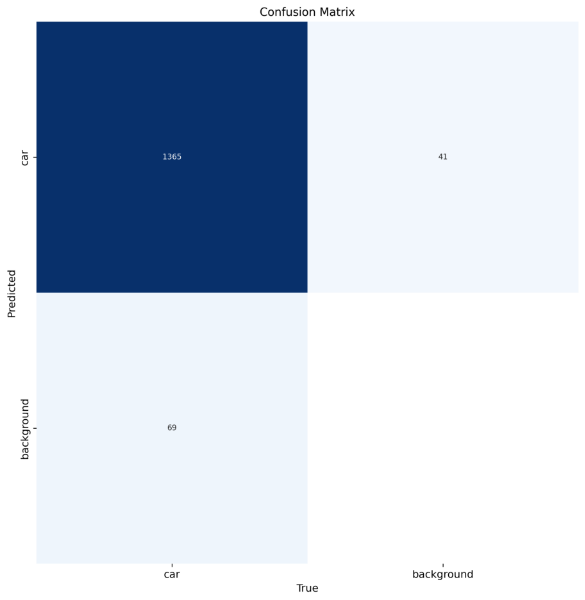
(c) Model #5 Precision-Recall Curve

Looking at the results above, we see that Model #1 has an mAP value of 0.961, Model #3 has an mAP value of 0.991, and Model #5 has an mAP value of 0.995. The graphs also demonstrate that Model #5's PR curve is much closer to the upper-right corner compared to Model #3 and Model #1, meaning that it is capable of achieving a point with both higher precision and recall values at the same time. These improvements once again clearly demonstrate that sim2real truly bridges the reality gap better than simulation data or purely real data alone.

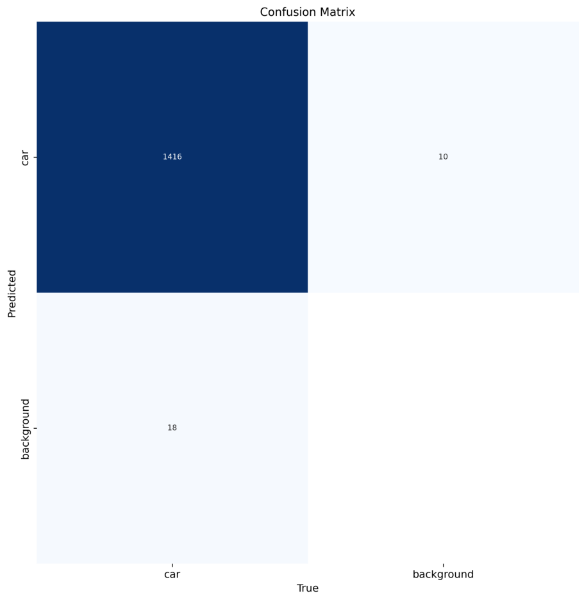
Going beyond the mere numbers above, it is important to further analyze exactly how many frames the models predicted correctly or incorrectly. Every single frame can mean the difference between continuing to race, or potentially colliding and wiping out fatally with another vehicle, resulting in millions of dollars of damages. Hence, in such safetycritical domains as high-speed autonomous racing, every single frame and detection matters. Considering this, the results comparing the three models are shown below:



(a) Model #1 Confusion Matrix



(b) Model #3 Confusion Matrix



(c) Model #5 Confusion Matrix

Analyzing the confusion matrices displayed above, we can further see that sim2real truly provides significant boosts in performance to our models. Starting with Model #1 (trained only on real data), it only correctly identifies 1298 of the test dataset frames, while incorrectly missing the vehicle in 136 frames, and inadvertently labelling a ghost car for 169 frames.

Beating this model, Model #3 (trained on simulation + real data) correctly identifies 1365 frames, while missing only 69 frames, and hallucinating a vehicle for 41 frames. Finally, the best model which stands out significantly above the rest is Model #5 (trained on sim2real + real data), which correctly identifies an impressive 1416 frames, while only missing a mere 18 frames, and accidentally seeing a vehicle briefly for a mere 10 frames.

Overall Quantitative Analysis

Thus, analyzing all the above results from a holistic point of view, it becomes undoubtedly clear the advancements and benefits that sim2real brings to the table, and the significant improvements in performance that it contributes to our models. Hence, this objectively proves that our sim2real approach truly is a success, and the numerical evidence completely justifies and supports this analysis.

7.5 Qualitative (Visual) Results

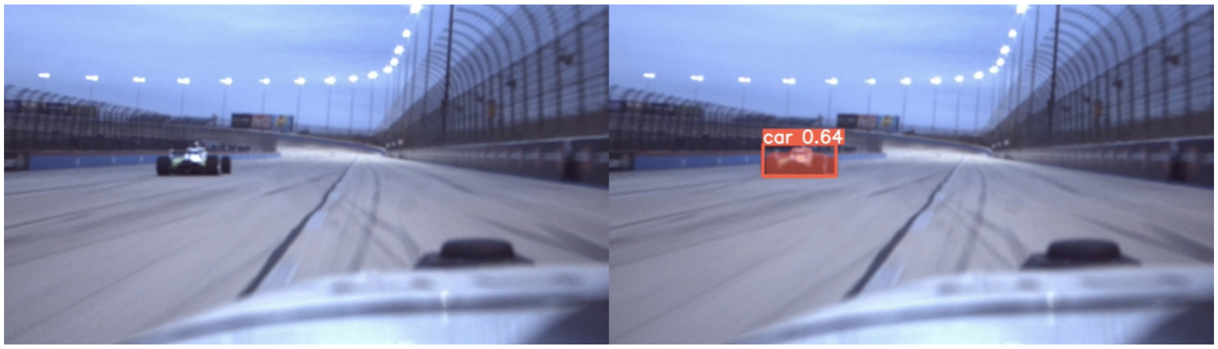
Going beyond numbers and statistics, it is often extremely useful to further visualize the results directly. As a result, we conduct the same experiments above, albeit this time by providing side-by-side tiled comparisons between the model output videos on the test dataset, synchronized across every individual frame for an objective comparison.

Model #2 (Pure Sim) vs. Model #4 (Pure Sim2Real)

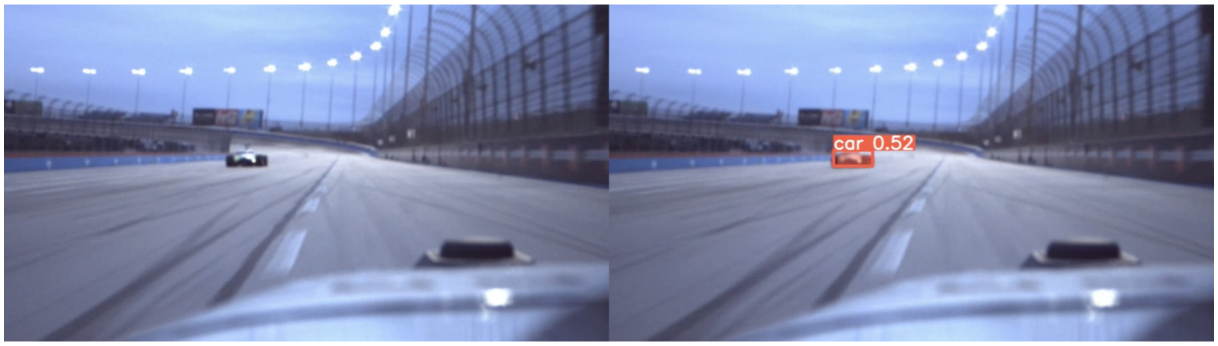
For context, the following format and convention will be used for this section:

- **Left Image** represents **Model #2 (Pure Sim)** output, referred to as **M2**
- **Right Image** represents **Model #4 (Pure Sim2Real)** output, referred to as **M4**

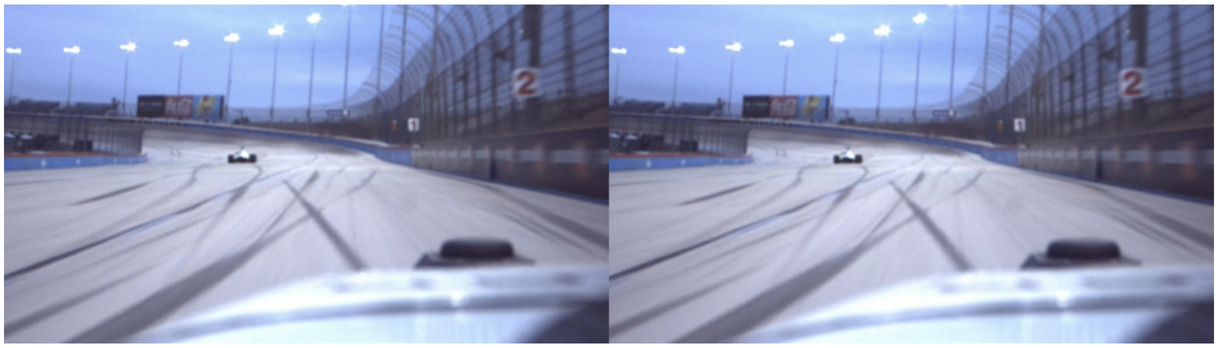
All images presented below will follow this exact format for consistency and comparison. Furthermore, all frames presented below are in sequential order, increasing in time steps from start to finish on the same test dataset previously used in the quantitative results.



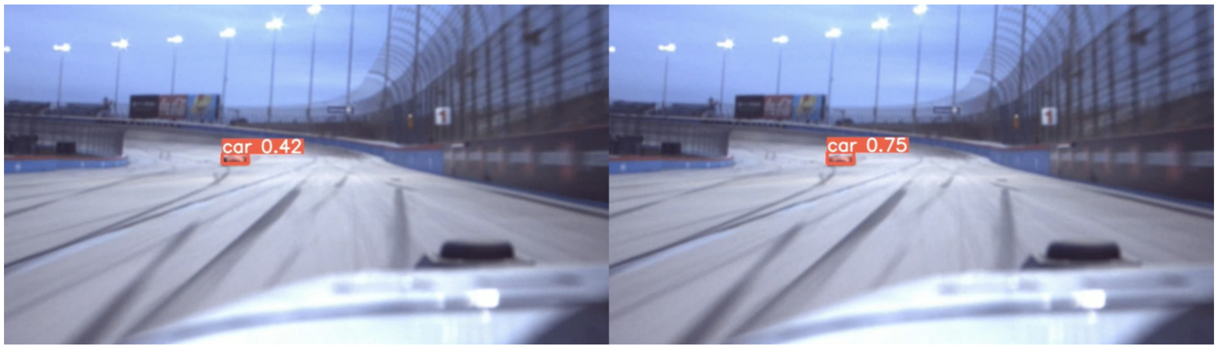
Starting off the test run, we can see right away that M4 outperforms M2 by detecting the race car almost instantly. Furthermore, this detection has high confidence of 0.64, and a filled out segmentation mask. M2 still doesn't recognize the vehicle.



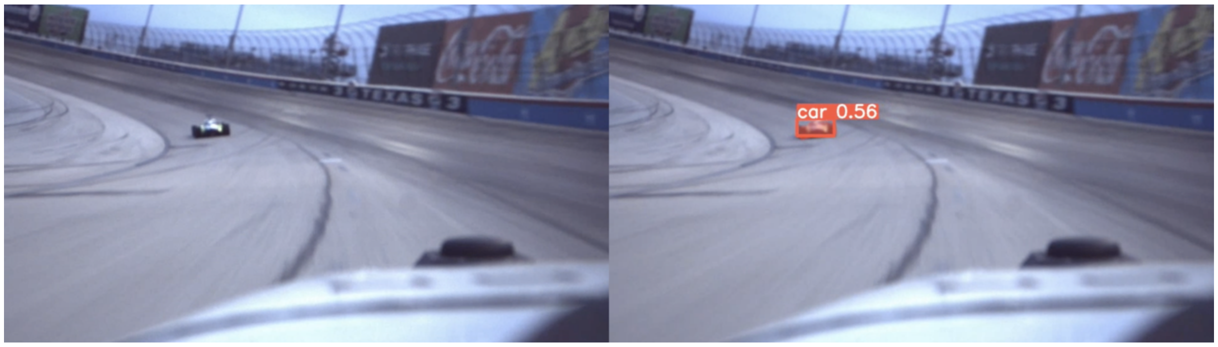
As the vehicles progress, M4's confidence level fluctuates a little, and drops down to 0.52. Nonetheless, it still outperforms M2 which still isn't able to detect the vehicle. Furthermore, M4's segmentation mask is still perfectly outlined as it was before.



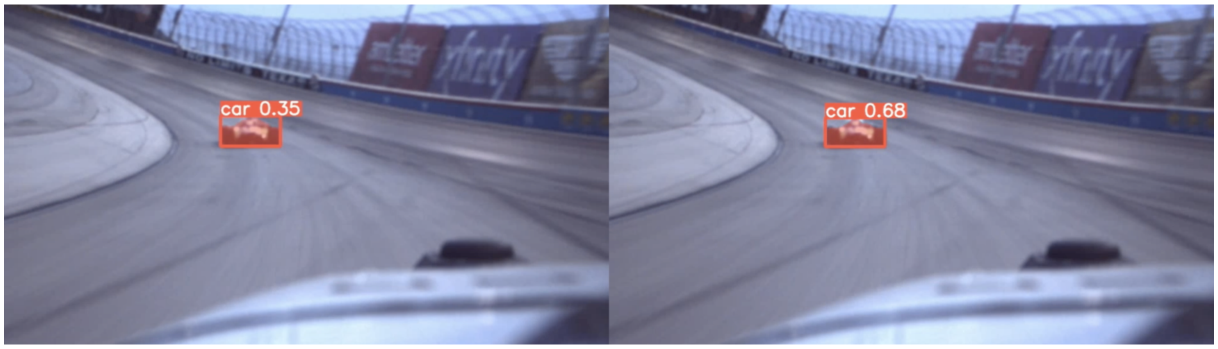
However, there are some occasional frames in which even M4 falters in the output detection, and is unable to identify the vehicle in frame. In this case, M2 continues to be unable to detect the race car as it did in previous frames.



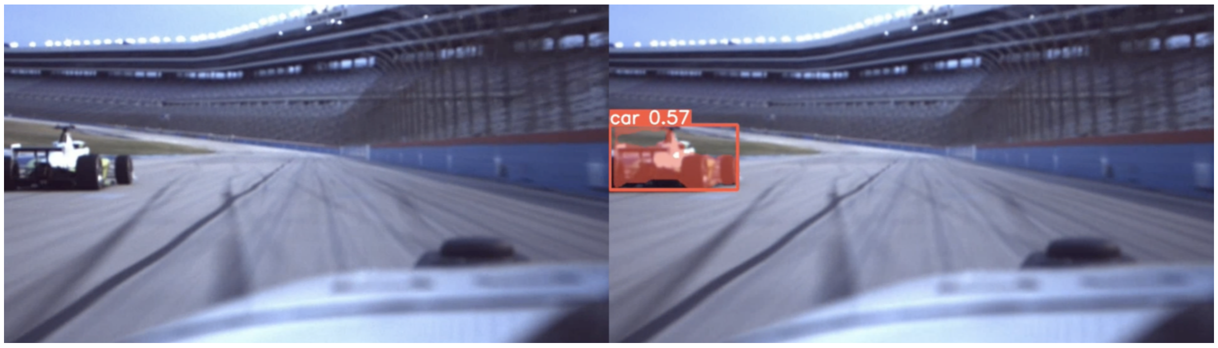
Eventually, M2 is finally able to detect the vehicle, and has a well segmented mask from far away. M4 continues to maintain its superior performance, with a similarly full segmentation of the race car, but with an additional higher confidence level of 0.75.



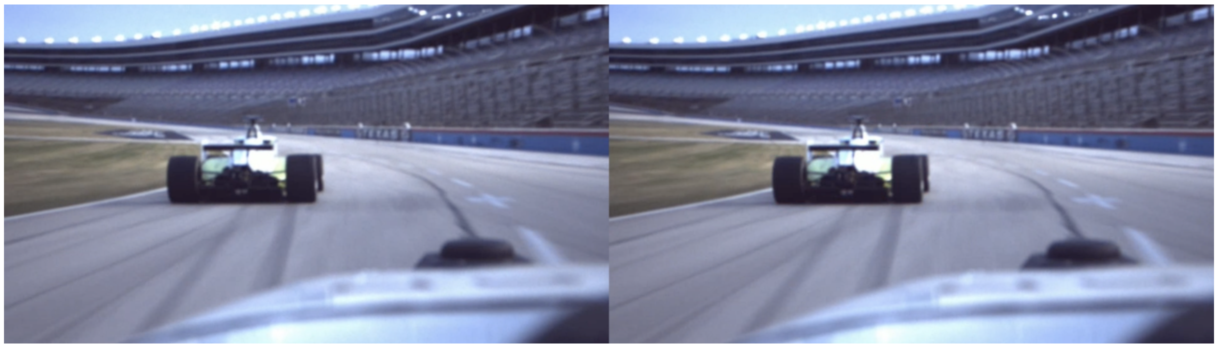
Soon however, M2 once again drops its ability to detect the race car ahead. Despite this, M4 continues to detect the vehicle cleanly, albeit with a lower confidence level of 0.56. The mask is still well segmented though in the image.



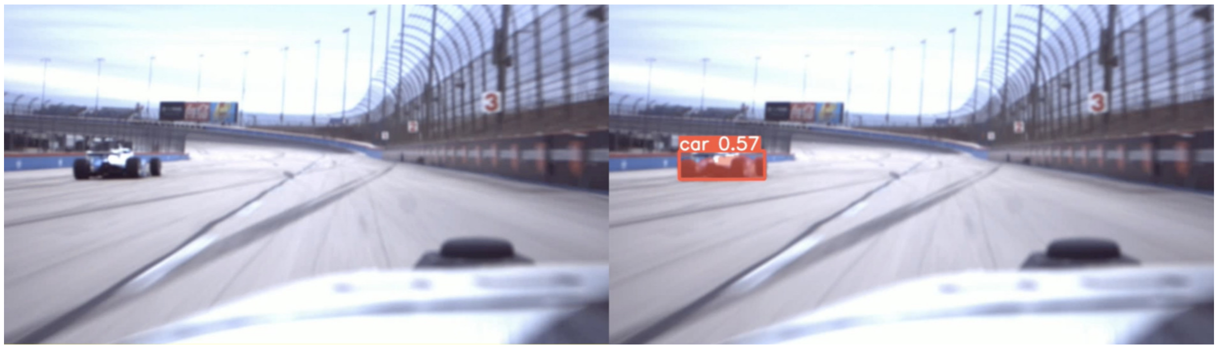
M2 once again is able to detect the race car. It has a very low confidence level of 0.35, significantly lower than M4's confidence of 0.68. Both segmentation masks fully encompass the entire vehicle. However, we can see how M2's performance fluctuates much more often and is volatile compared to M4.



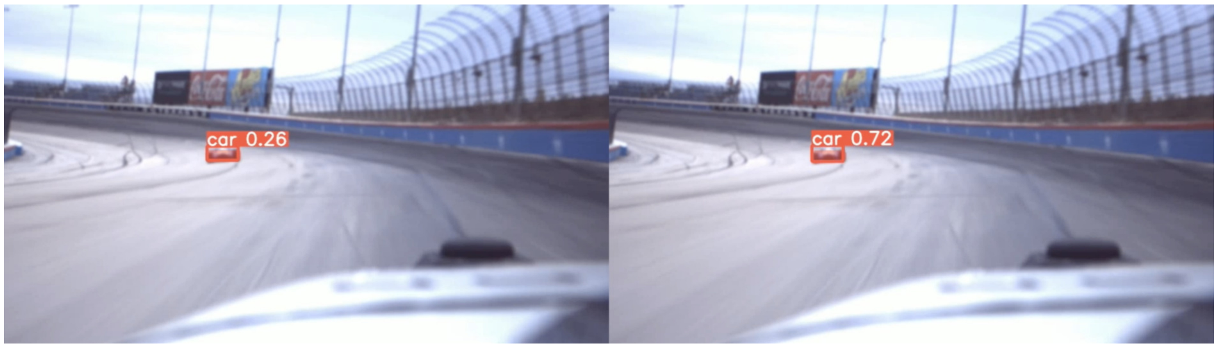
M4 is successfully able to identify and segment the race car. The segmentation mask is a little degraded in a few pixels on the top right, but nonetheless it detects the vehicle with average confidence of 0.57, while M2 is unable to detect the vehicle entirely.



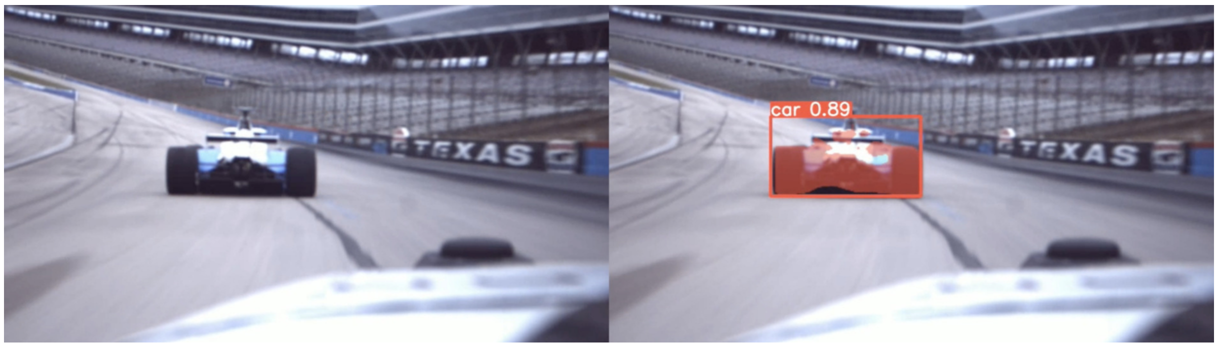
Once again, we can see the M4 model trained purely on sim2real isn't perfect, and misses detection of the vehicle. However, in this frame, M2 isn't able to detect the vehicle as well, so neither model is able to see the race car ahead.



As the second half of the test dataset run begins, we can see M4 once again is the first model to detect the race car ahead, with a confidence level of 0.57. The segmentation mask is a little degraded on the top of the vehicle, but overall looks pretty good, and surpasses M2 which is still unable to detect the vehicle entirely.



Soon after, both models M2 and M4 are able to successfully detect the race car in view ahead. However, M2 has a very low confidence level of 0.26, while M4 has a significantly higher confidence of 0.76. Nonetheless, both models have very clean segmented masks of the race car, and perform well in this regard.



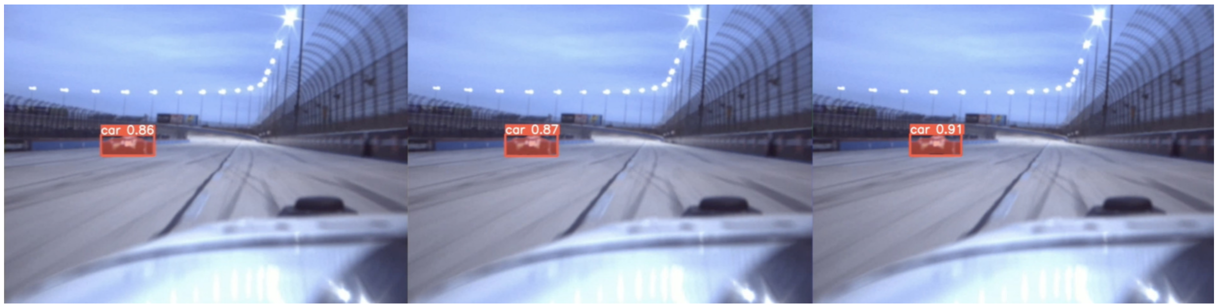
Finally as the test dataset video comes near the end, the race car approaches much closer to the chase vehicle. Unsurprisingly, M2 is unable to detect the car once again, while M4 is able to detect with an extremely high confidence of 0.89. However, the segmentation mask is extremely poor, and has major gaps and deterioration spreading from the center of the vehicle towards the top-right.

Model #1 (Real) vs. Model #3 (Sim + Real) vs. Model #5 (Sim2Real + Real)

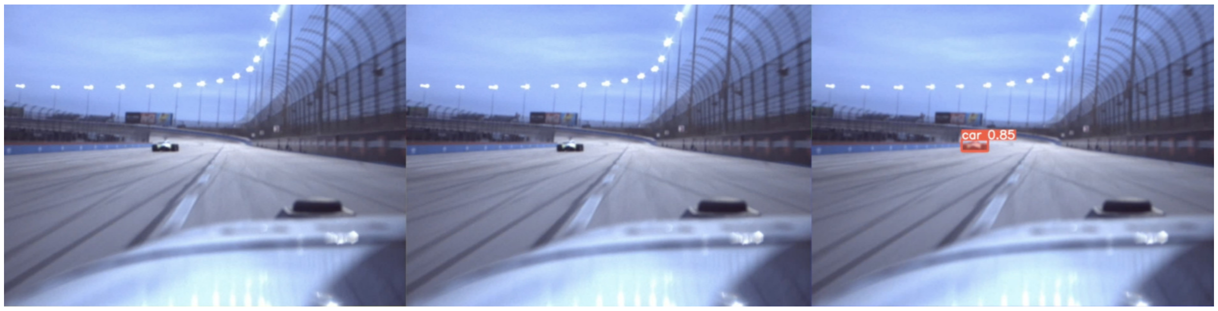
For context, the following format and convention will be used for this section:

- **Left Image** represents **Model #1 (Pure Real)** output, referred to as **M1**
- **Middle Image** represents **Model #3 (Sim + Real)** output, referred to as **M3**
- **Right Image** represents **Model #5 (Sim2Real + Real)** output, referred to as **M5**

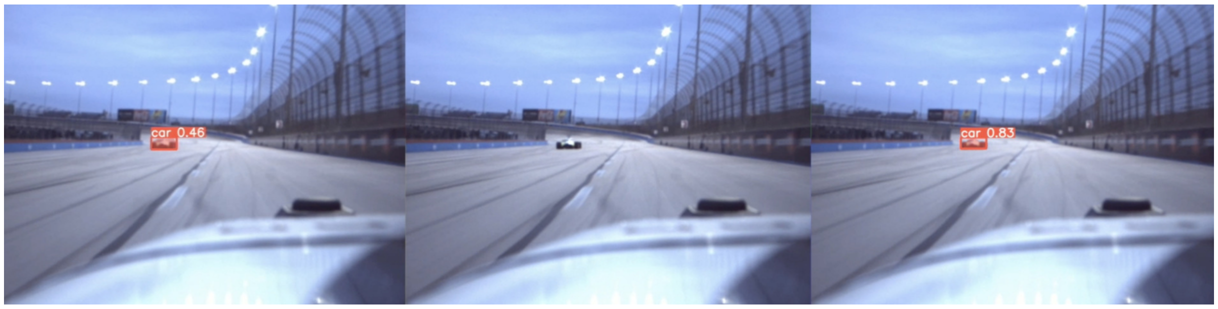
All images presented below will follow this exact format for consistency and comparison. Furthermore, all frames presented below are in sequential order, increasing in time steps from start to finish on the same test dataset previously used in the quantitative results.



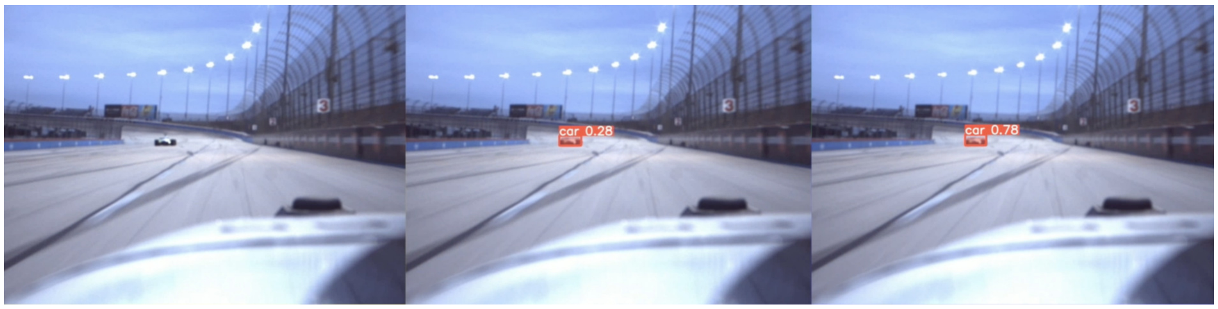
Starting off the run in the test dataset, we can see that all three models M1, M3, and M5 successfully detect the vehicle, which is a great start! However, it stands out that M5 has the highest confidence level of 0.91, while M3 and M1 trail behind at 0.87 and 0.86 respectively. All three masks are perfectly segmented and cover the entire vehicle.



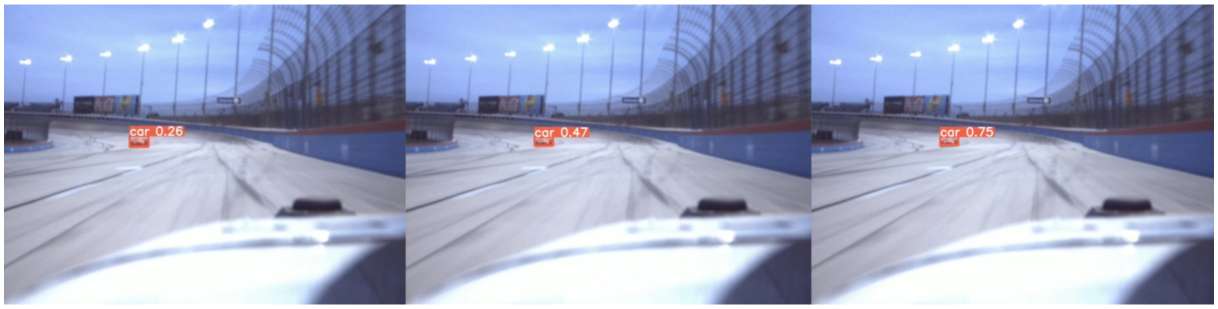
Very soon after, however, both M1 and M3 lose their ability to detect the race car ahead in view. Considering this, M5 shines above the rest and truly stands out, as it not only continues to detect the vehicle ahead, but further does so with a very high confidence of 0.85, and a perfect segmentation mask fully encompassing the entire vehicle.



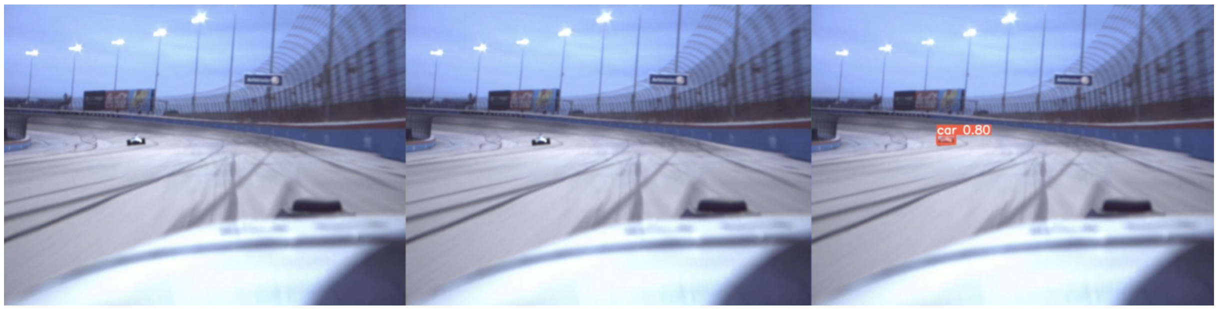
A few frames later, M1 regains detection of the vehicle in view, but does so with a very poor confidence level of 0.46. M3 still remains unable to detect the vehicle as it did before. M5 continues to stay on top, maintaining a very high confidence of 0.83 and perfect segmentation mask as well.



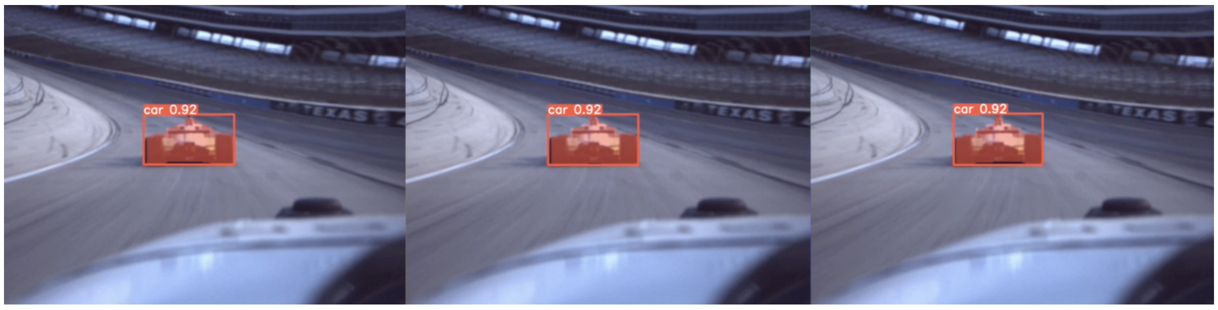
Thereafter, M1 once again fluctuates and loses identification of the race car. M3 however is now able to detect the vehicle, albeit with an extremely poor confidence of only mere 0.28. M5 yet again outperforms both models, maintaining a high confidence level of 0.78 and perfect segmentation.



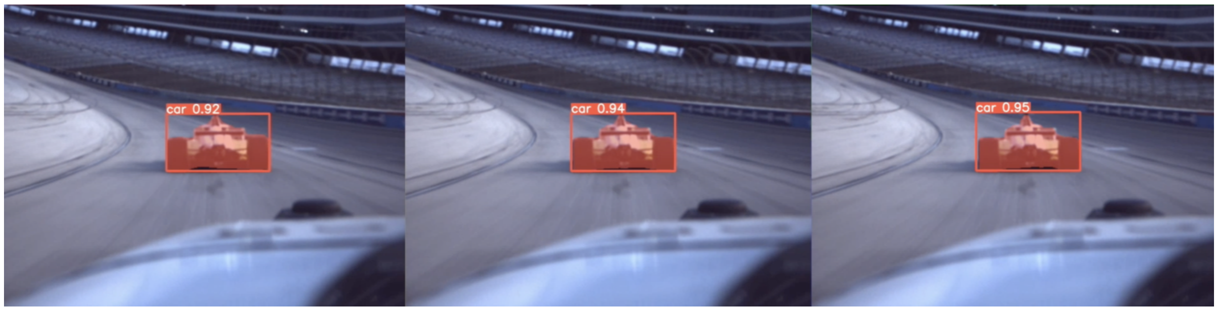
Soon after, both M1 and M3 once again demonstrate the ability to recognize the race car, while M5 continues to detect it as before. Nonetheless despite all three vehicles being able to identify the car, they do so in very different confidence levels. M1 has extremely poor confidence of 0.26, M3 is slightly better with a subpar confidence of 0.47, and M5 outshines the rest with a high confidence of 0.75. All three segmentation masks are well-defined and cover the vehicle.



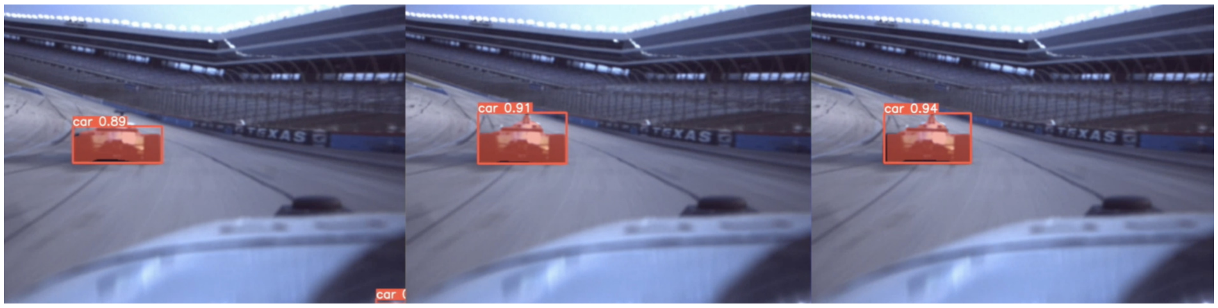
Unsurprisingly, both M1 and M3 are yet again unable to detect the race car in view, while M5 remains completely unfazed. Furthermore, it has a very high confidence of 0.80, and maintains its perfect segmentation mask as it has done consistently throughout the run thus far.



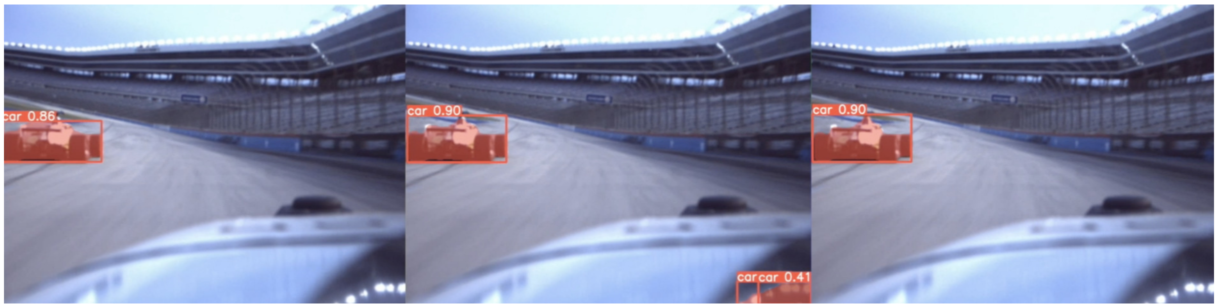
As we jump further into the race, the vehicle ahead comes much closer to the chase vehicle. In this close-up detection, all three models are able to successfully detect and segment the vehicle in identical ways, with identical confidence levels of 0.92 as well.



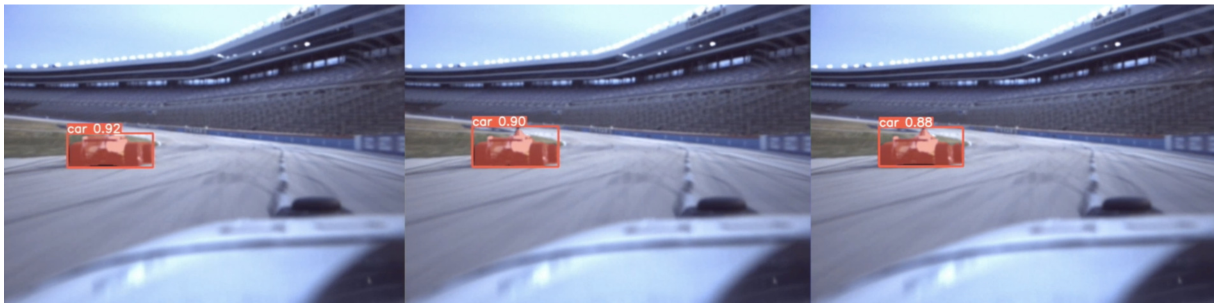
While the perfect segmentation masks are maintained across all three models, soon after we can see that M5 still edges out on top with an exceptionally high confidence of 0.95, while M3 and M1 have slightly lower confidence levels of 0.94 and 0.92.



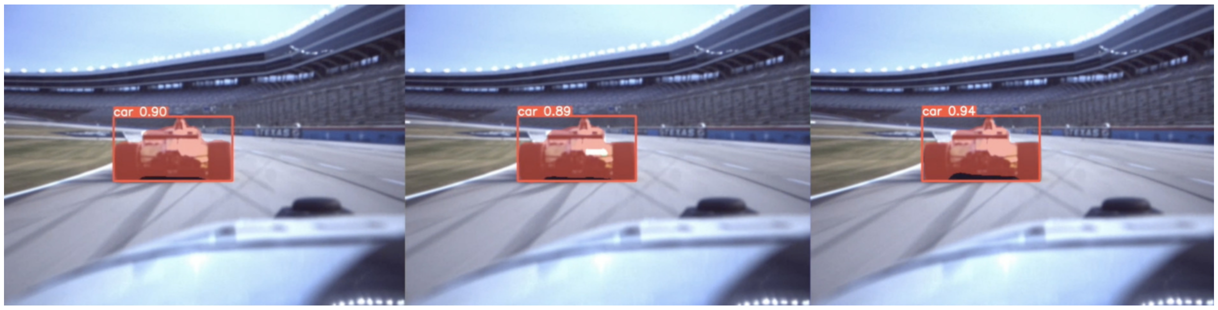
In this frame, all three models continue to identify and segment the vehicle. However, M1 incorrectly segments and loses the upper half of the race car. It additionally also has a false detection on the bottom right of the image. M3 and M5 still have good detection and confidence, but M5 is still the highest with an exceptional 0.94.



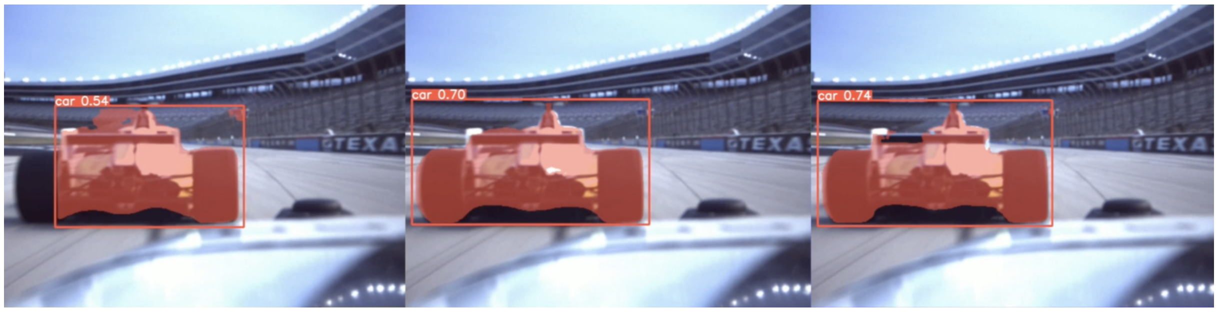
As the frames progress, M1 has the lowest confidence of 0.86, but at least properly segments the race car. Both M3 and M5 perfectly segment and maintain a high confidence of 0.90, but a key difference is that M3 now appears to have a false detection on the bottom right of the image. M5 remains clean and consistent in its performance.



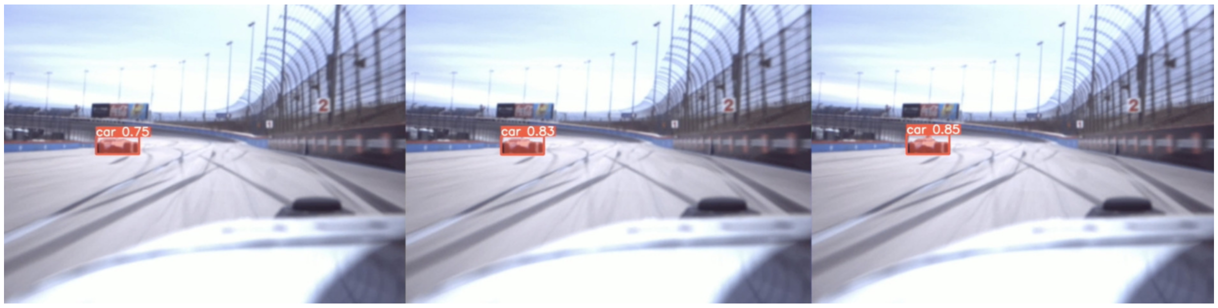
In a very rare few frames, we can see that occasionally M1 outperforms the other models with a confidence of 0.92, followed shortly behind by M3 with 0.90 and M5 with 0.88. However, these frames are extremely rare, demonstrating that M5 still performs significantly better than the rest across the test run thusfar.



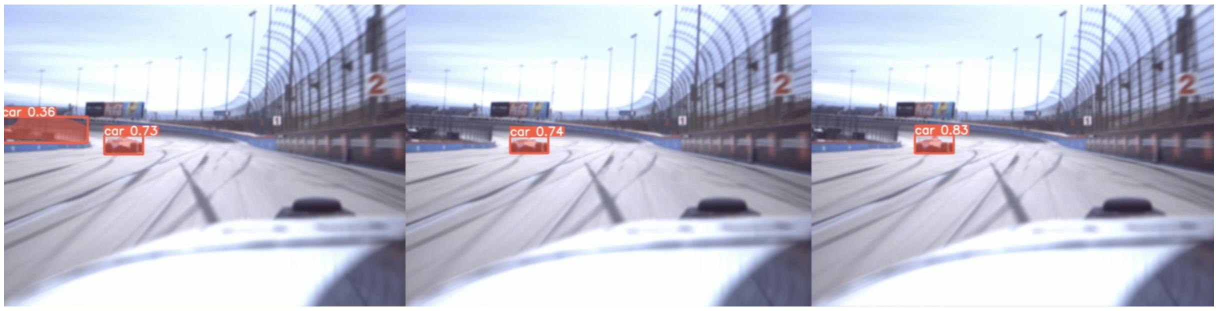
A few frames later, we can confirm that the previously mentioned statement is true. M5 maintains the highest confidence of 0.94 over the other two models. It also has a perfect segmentation, while M3 has a missing degraded segmentation patch in the center of the car, and M1 has lower confidence.



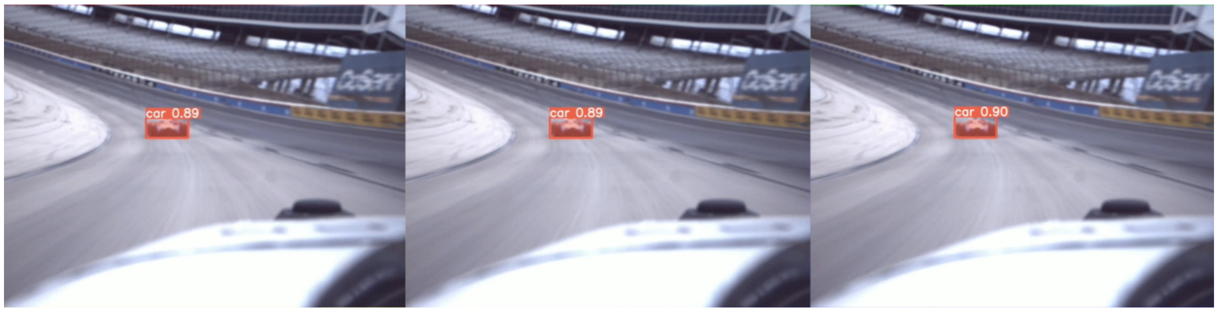
As the race car approaches very close to the chase vehicle, we can yet again see another benefit of M5's performance over the other models. M1 has poor detection and segmentation, missing a major part of the left side of the vehicle, and erroneously segmenting some of the background as the race car. M3 is slightly better, but still has a few degraded pixels in the segmentation. M5 remains unfazed, with the highest confidence of 0.74, and a near-perfect segmentation of the entire race car in view.



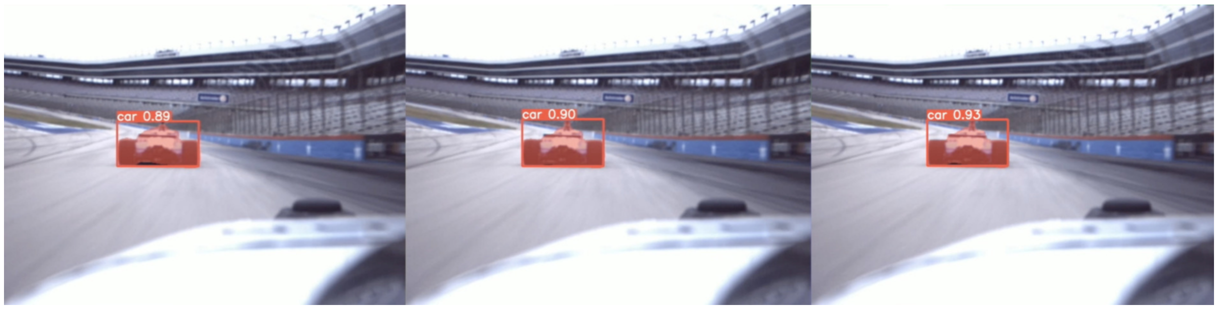
As we enter the second half of the test data, we can see all three models performing well and segmenting the race car ahead. M5 has the highest confidence of 0.85, followed by M3 and M1 with 0.83 and 0.75 respectively. Although the differences may seem slight, in the full context of the run, M5 demonstrates consistent superior performance overall.



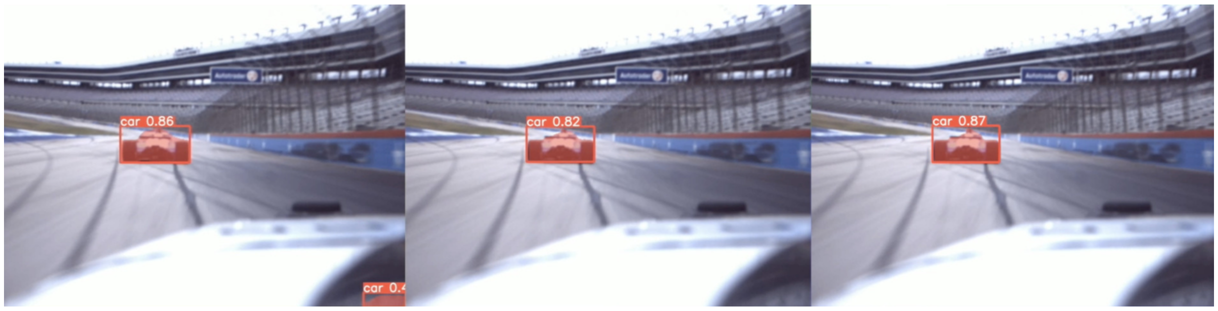
M1 has more false detections, erroneously identifying some of the background track as a race car. Nonetheless, when observing the real vehicle ahead, all three models successfully segment the full shape, and M5 has the highest confidence of 0.83, much higher than the other two models of 0.73 and 0.74.



As the run continues, we can see all three models performing well together. Despite this, M5 never ceases to lose its edge over the other models, with a confidence level 0.01 over both other models. Although slight, it still demonstrates superior performance nonetheless.



As the run continues, we can see M5 widen the confidence gap, with an exceptional confidence level of 0.93, trailing the other two models with confidence of 0.90 and 0.89 for M3 and M1 respectively. Furthermore, we can observe that M1's segmentation mask is a little less refined with fuzzy edges bleeding off the sides of the race car. M3 is slightly better, but still has a few pixels on the top right where it goes over the edge. M5 objectively stands out with the best segmentation, and a perfect outline of the race car in view.



Finally in the last frame from the run, we can see M1 have yet another false detection on the bottom-right corner of the image, which diminishes the value of the model despite a 0.86 confidence detection of the actual car. M3 has no false detection, but a lower confidence of 0.82. Finally, M5 unsurprisingly stands out above the rest, with the highest confidence of 0.87, a perfect segmentation mask, and no false detections in the frame.

Overall Qualitative Analysis

Ultimately, looking at the visual results above, there exists no doubt that sim2real has proven its capabilities and benefits. Throughout the comparisons, sim2real has demonstrated its ability to bridge the reality gap and improve performance compared to simulation data, and this increased accuracy of detection and segmentation demonstrates that the sim2real converted images in training can realistically match real-world images for the model. Furthermore, sim2real demonstrates the greatest amount of consistency in performance as well, with less fluctuations and volatility compared to the other models. This is especially useful and essential for our autonomous racing use-case, as consistency in our detection and segmentation outputs will provide the most stable information for other teams to utilize for smooth path planning and crucial decision making during the race.

Hence, the overall results look amazing and demonstrate meaningful improvements, and we have thus proven that our zero-shot sim2real approach is a success!

Chapter 8

Conclusion and Future Works

In summary, the work that was conducted through this project ultimately serves to improve our self-driving race car's performance on the race track in the real world. Through the latest advancements and techniques in photorealistic style transfer, we carefully considered various approaches, and ultimately modified, implemented, and tailored a model to our specific use-case. The results demonstrate significantly meaningful and tangible improvements to our instance segmentation perception algorithms, and have been accepted and commended by the entire team. This novel sim2real image generation pipeline will be used in all future upcoming races for AI Racing Tech's vehicle, and will be presented as an open resource to the Indy Autonomous Challenge racing community through future workshops, presentations, and more.

The invaluable contribution of this project lies in the ability to confidently state that the sim2real model truly generates realistic outputs to train our vehicles on. Furthermore, the key benefit comes from achieving this through a zero-shot approach, due to the ability to use real-world images even if they differ from simulation inputs by the number of vehicles in view, race track trained on, lighting conditions, etc. As a result, we have unlocked the potential to generate and train on thousands of images of simulated racing scenarios, covering all the edge cases of collisions, crashes, high density races with multiple vehicles, racing in brand new race tracks regardless of location around the world, etc.

In the future, the team aims to continue developing this project further, experimenting and modifying the techniques in order to generate even better sim2real generated outputs, and improve our data quality to maximize our instance segmentation models. In addition, this project further helped set up the full end-to-end pipeline from simulation to real-world vehicle, and as a result of the modular design and codebase, will allow immense flexibility to swap out future novel sim2real techniques as they become released and are optimized to run feasibly on our hardware.

The future potential for this project is truly limitless, and we are excited to see how the AI Racing Tech team continues to evolve alongside the Indy Autonomous Challenge for decades to come. When it comes to racing, the track may be the limit in a practical sense, but only the sky is truly the limit for innovation.

Bibliography

[1] Kfir Aberman et al. “Neural Best-Buddies: Sparse Cross-Domain Correspondence”. In: *ACM Transactions on Graphics* 37.4 (Aug. 31, 2018), pp. 1–14. arXiv: `1805.04140[cs]`.

[2] *CVAT*. URL: `https://www.cvat.ai/`.

[3] Yingying Deng et al. *StyTr^2: Image Style Transfer with Transformers*. Apr. 1, 2022. arXiv: `2105.14576[cs,eess]`.

[4] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. June 3, 2021. arXiv: `2010.11929[cs]`.

[5] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. “Image Style Transfer Using Convolutional Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. June 2016, pp. 2414–2423.

[6] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. Dec. 16, 2020. arXiv: `2006.11239[cs,stat]`.

[7] *Indy Autonomous Challenge*. Indy Autonomous Challenge. URL: `https://www.indyautonomouschal`
`com/challenge-about`.

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. Mar. 26, 2016. arXiv: `1603.08155[cs]`.

[9] Sunwoo Kim, Soohyun Kim, and Seungryong Kim. *Deep Translation Prior: Test-time Training for Photorealistic Style Transfer*. Jan. 17, 2022. arXiv: `2112.06150[cs]`.

[10] Alexander Kirillov et al. *Segment Anything*. Apr. 5, 2023. arXiv: `2304.02643[cs]`.

[11] Chuan Li and Michael Wand. *Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis*. Jan. 18, 2016. arXiv: `1601.04589[cs]`.

[12] Xueting Li et al. *Learning Linear Transformations for Fast Arbitrary Style Transfer*. Aug. 14, 2018. arXiv: `1808.04537[cs]`.

[13] Yijun Li et al. *A Closed-form Solution to Photorealistic Image Stylization*. July 26, 2018. arXiv: `1802.06474[cs]`.

[14] Tianwei Lin et al. *Drafting and Revision: Laplacian Pyramid Network for Fast High-Quality Artistic Style Transfer*. Apr. 17, 2021. arXiv: `2104.05376[cs,eess]`.

[15] Songhua Liu et al. *AdaAttN: Revisit Attention Mechanism in Arbitrary Neural Style Transfer*. Aug. 11, 2021. arXiv: `2108.03647[cs]`.

[16] Fujun Luan et al. *Deep Photo Style Transfer*. Apr. 10, 2017. arXiv: `1703.07511[cs]`.

[17] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. May 9, 2016. arXiv: `1506.02640[cs]`.

[18] Jascha Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. Nov. 18, 2015. arXiv: `1503.03585[cond-mat,q-bio,stat]`.

[19] Narek Tumanyan et al. *Splicing ViT Features for Semantic Appearance Transfer*. Jan. 2, 2022. arXiv: `2201.00424[cs]`.

[20] Ultralytics. *Home*. URL: `https://docs.ultralytics.com/`.

[21] Ashish Vaswani et al. *Attention Is All You Need*. Aug. 1, 2023. arXiv: `1706.03762[cs]`.

[22] Pei Wang, Yijun Li, and Nuno Vasconcelos. *Rethinking and Improving the Robustness of Image Style Transfer*. Apr. 7, 2021. arXiv: `2104.05623[cs,eess]`.

[23] Zhizhong Wang, Lei Zhao, and Wei Xing. *StyleDiffusion: Controllable Disentangled Style Transfer via Diffusion Models*. Aug. 15, 2023. arXiv: `2308.07863[cs]`.

[24] Zhizhong Wang et al. "GLStyleNet: exquisite style transfer combining global and local pyramid features". In: *IET Computer Vision* 14.8 (2020). _eprint: https://onlinelibrary.wiley.com/doi/ cvi.2019.0844, pp. 575–586.

[25] Jaejun Yoo et al. *Photorealistic Style Transfer via Wavelet Transforms*. Sept. 29, 2019. arXiv: `1903.09760[cs]`.

[26] Yuxin Zhang et al. *Inversion-Based Style Transfer with Diffusion Models*. Mar. 20, 2023. arXiv: `2211.13203[cs]`.