FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generating Synthetic Training Data for Video Surveillance Applications

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Johannes Eschner
Matrikelnummer 01633402

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Werner Kloihofer
Adriano Souza Ribeiro

Wien, 9. Oktober 2019

Johannes Eschner	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Generating Synthetic Training Data for Video Surveillance Applications

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Johannes Eschner
Registration Number 01633402

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Werner Kloihofer
Adriano Souza Ribeiro

Vienna, 9th October, 2019

Johannes Eschner | Michael Wimmer

Erklärung zur Verfassung der Arbeit

Johannes Eschner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 9. Oktober 2019

Johannes Eschner

Danksagung

Ich möchte mich bei Werner Kloihofer und Adriano Souza Ribiero für ihre Unterstützung während des gesamten Projektes bedanken. Sie haben diese Arbeit mit ihrem wertvollen Input während der Recherche und Implementierung bereichert und haben mir bei der Datenakquisition sowie bei der Evaluierung geholfen. Ich möchte mich auch bei meinem Betreuer Michael Wimmer bedanken, der mich mit ausführlichem Feedback und konstruktiven Kritiken im Schreibprozess dieser Arbeit unterstützt hat. Zusätzlich möchte ich mich auch bei Florian Rudolf bedanken, der den initialen Kontakt zum Industriepartner geknüpft hat, mit dem ich im Rahmen dieser Arbeit kooperiert habe.

Abschließend möchte ich mich auch bei meinen Eltern dafür bedanken, dass sie mein Studium ermöglicht haben sowie bei allem Familienmitgliedern, Freunden und Kollegen, die mich während meiner Arbeit unterstützt haben.

Die Aktivitäten, die hinter diesem Projekt stehen und die Personen, die an der Betreuung dieser Arbeit beteiligt sind, wurden im Rahmen der Finanzhilfevereinbarung 780788 aus dem Horizon 2020-Projekt ALOHA der Europäischen Union finanziert.

Acknowledgements

I want to thank Werner Kloihofer and Adriano Souza Ribiero for their support throughout the whole project. They both provided valuable inputs during the research and the implementation and helped me with the data acquisition as well as the evaluation. I also want to thank my supervisor Michael Wimmer for providing me with support, feedback and constructive criticism during the writing of this thesis. Additionally, I want to thank Florian Rudolf, who provided the initial contact to the industry partner I worked with for this thesis.

Finally, I also want to thank my parents for enabling my studies as well as all the family members, friends and colleagues who supported me during the work.

The activities behind this project and the people involved in supervising the thesis have received funding for their studies from the European Union's Horizon 2020 project ALOHA, under grant agreement 780788.

Kurzfassung

Mit der wachsenden Nachfrage nach immer umfangreicheren Computer Vision Systemen steigt in den letzten Jahren auch der Bedarf an gelabelten Ground-Truth Datensätzen für solche Systeme. Diese Datensätze werden zur Evaluierung und zum Training von Computer Vision Algorithmen genutzt und werden normalerweise durch manuelles Annotieren von semantischen Labels in Bildern oder Bildsequenzen erzeugt. Das Generieren von synthetischen Videos bietet einen alternativen Ansatz für das Annotieren von Bilddaten. Hier können Labels und Bildsequenzen mittels einer 3D-Renderengine gleichzeitig erzeugt werden. Viele bestehende Frameworks zur Erzeugung von synthetischen Datensätzen konzentrieren sich auf den Bereich des autonomen Fahrens, wo große Mengen an gelabelten Daten benötigt werden.

In dieser Arbeit wird eine Implementierung eines Frameworks zur Erzeugung synthetischer Daten für die Evaluierung von Trackingalgorithmen aus dem Bereich der Videoüberwachung präsentiert. Dieses Framework nutzt eine kommerziell erhältliche Game Engine als Renderer zur Erzeugung von synthetischen Videoclips, welche verschiedene Szenarien aus dem Bereich der Videoüberwachung darstellen. Diese Szenarien beinhalten eine Vielzahl an Interaktionen verschiedener Charaktere in einer virtuell nachgebauten Umgebung. Eine Sammlung solcher Clips wird dann mit echten Videos vergleichen, indem sie als Inputs für zwei Trackingalgorithmen genutzt werden. Während die Erzeugung von synthetische Daten in Echtzeit mit einer Game Engine weniger arbeitsintensiv als manuelles Annotieren ist, sind die Trackingergebnisse der echten Daten bei beiden getesteten Algorithmen besser. Das deutet darauf hin, dass die synthetischen Daten aus dem Framework nur bedingt zur Evaluierung von Trackingalgorithmen geeignet sind.

Abstract

As the demand for ever-more capable computer vision systems has been increasing in recent years, there is a growing need for labeled ground-truth data for such systems. These ground-truth datasets are used for the training and evaluation of computer vision algorithms and are usually created by manually annotating images or image sequences with semantic labels. Synthetic video generation provides an alternative approach to the problem of generating labels. Here, the label data and the image sequences can be created simultaneously by utilizing a 3D render engine. Many of the existing frameworks for generating such synthetic datasets focus the context of autonomous driving, where vast amounts of labeled input data are needed.

In this thesis an implementation of a synthetic data generation framework for evaluating tracking algorithms in the context of video surveillance is presented. This framework uses a commercially available game engine as a renderer to generate synthetic video clips that depict different scenarios that can occur in a video surveillance setting. These scenarios include a multitude of interactions of different characters in a reconstructed environment. A collection of such synthetic clips is then compared to real videos by using it as an input for two different tracking algorithms. While producing synthetic ground-truth data in real time using a game engine is less work intensive than manual annotation, the results of the evaluation show that both tracking algorithms perform better on real data. This suggests that the synthetic data coming from the framework is limited in its suitability for evaluating tracking algorithms.

Contents

CHAPTER 1

Introduction

With the ever-increasing use of computer vision systems in fields ranging from document analysis to autonomous vehicles, there is a growing demand for ways to test and train such systems. Usually, a computer vision system is evaluated by comparing its performance to some sort of ground-truth dataset. In the case of an image-segmentation algorithm, such a ground-truth dataset could for example be a set of images where all pixels belonging to one object are marked in the same color. This data can then be compared to the results the algorithm delivers, to see how well the segmentation was performed by the algorithm.

Creating new ground-truth datasets is a labor-intensive task since there has to be a human in the loop to generate the data. For example, the creation of the Microsoft COCO dataset, which contains 2.5 million labeled instances in 328,000 images, took over 70,000 worker hours [LMB+14]. This problem only escalates when it comes to labeled video data, where continuity between frames also has to be taken into account. As the cost of creating new datasets is very high, many developers of computer vision systems rely on ready-to-use publicly available ground-truth datasets for training and evaluation. Utilizing such datasets comes with the advantage that the performance of different algorithms can easily be benchmarked by comparing the results they deliver on the same dataset. However, in many cases there are simply no datasets available that fit the needs of a certain computer vision system. In such cases there is no other choice than to create a custom dataset.

Generating synthetic ground-truth data is a possible solution to the hardship of creating new datasets. Here, instead of real images, computer-generated renderings are used to train and test a computer vision system. In the end these systems are meant to work on real images, so it is of importance that the synthetic dataset does not deviate too far from the real-world representations.

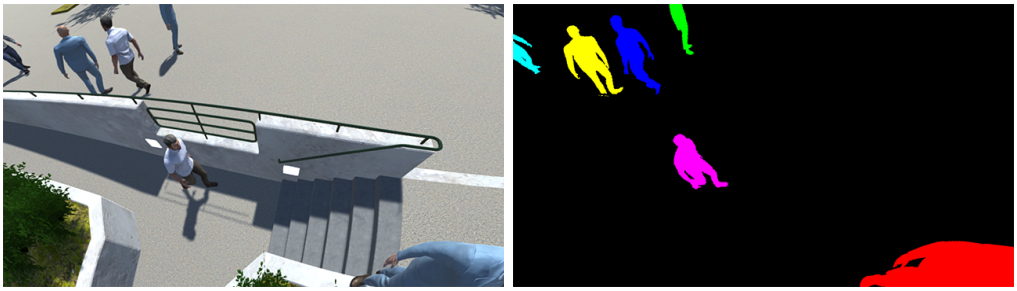
With the advent of nearly photo-realistic rendering, generating synthetic datasets became possible for a wide range of applications. Although creating computer-generated images

comes with the upfront cost of recreating a real-world scenario beforehand, it comes with the big advantage that creating labels is basically free. All objects in a scene are already known to the computer that renders the image, so ground-truth labels can be created on the fly.

Apart from being a possibly more cost-effective way of creating ground-truth datasets, synthetic data generation also allows more flexible use cases. If, for example, it is deemed necessary to see the same scene from a different angle or at different time of day, such changes can quickly be implemented in a computer-generated scene. Another advantage is that synthetic data can be created for use cases where real data is difficult to obtain, for example for security or data protection reasons.

1.1 Aim of this thesis

The aim of this thesis is to take a look at how the game engine Unity [1] can be utilized to generate a synthetic ground-truth dataset for a video surveillance system. This dataset consists of video clips of simulated real-word scenarios and corresponding labels which can be used for the training and evaluation of tracking algorithms in the context of video surveillance. For the scope of this thesis, however, the focus lies solely on the evaluation aspect, as the generated dataset is used to evaluate two tracking algorithms. Part of this evaluation is a comparison of tracking results from real and synthetic video inputs to see whether the synthtetic data is suitable for evaluation tracker performance. A sample frame with corresponding labels is depicted in Figure 1.1.



(a) Generated video frame. (b) Labels for the frame.

Figure 1.1: A sample from the generated dataset.

Apart from creating the dataset itself, we also create a reusable framework that allows for more general synthetic data generation within the Unity editor. This framework focuses on generating videos of human interactions with an environment and providing label data by utilizing the built-in real-time render engine of Unity. For this purpose, a collection of assets for constructing new scenarios and a Unity plug-in for managing the data generation is provided. The framework itself consists of a custom Unity editor and

[1] `https://unity.com/` Accessed: 10.09.2019

an interface that allows direct user interaction. Via this editor, character paths can be defined using waypoints. Trigger objects and cameras can also be set up by the user to simulate scenarios from the context of video surveillance.

Another contribution of this thesis is the evaluation of the feasibility of using Unity as an out-of-the-box solution for synthetic data generation. Here, we evaluate to what extent the tool-set of Unity provides a convenient basis for generating synthetic datasets. Part of this evaluation of convenience is to try to use as many built-in features of the engine as possible to reduce the need of creating and maintaining custom-made solutions. This is not only true for scripts related to post-processing but also for assets, where we try to source as many as possible from existing packages via the Unity Asset Store.

CHAPTER 2

Related Work

With the high cost associated with sourcing high quality ground-truth data for computer vision systems there have been numerous different approaches to creating computer-generated ground-truth datasets. In this chapter we take a look at different approaches and their use cases, as well as research on possible downsides of using synthetic data in place of real, annotated videos.

Most research on synthetic ground-truth data is done in the context of autonomous driving, where large amounts of annotated data are required for training and evaluation purposes. The datasets in this area of research are mainly for semantic segmentation algorithms that are used for object detection.

Richter et al. [RVRK16] demonstrate that the highly realistic graphical output from off-the-shelf closed-source video games can be used to create labeled training data. For this approach they intercept the communication between the game Grand Theft Auto V and the graphics hardware to extract pixel labels for areas that share the same mesh, texture and shader. Each of these areas gets a unique signature that is persistent over all recordings. The semantic information for these labels is however not obtained automatically, as there is no direct access to the game itself. Manual work is required to combine multiple object signatures to one object and to classify them. Once an object has been set up, the labels are automatically propagated through the following frames, greatly reducing the manual labeling effort. While this approach greatly reduces the workload for generating semantic segmentation ground-truth data, it is also rather inflexible as the possible scenarios are limited by the capabilities of the game. It also does not provide a fully automatic way of synthetic data generation as the labels have to be classified by hand.

The URSA dataset by Angus et al. [AEK+18] improves on the approach of [RVRK16] by uniquely identifying all objects of interest beforehand and then injecting textures when the object is requested by the engine. All unique objects are then manually classified,

with the advantage that once the annotation is determined for a unique object, it can be recycled throughout the game. This gives the system better flexibility than the per-frame annotation in [RVRK16], but it does still require a significant amount of manual work.

A third approach to generating synthetic semantic segmentation data is presented by Ros et al. [RSM+16] in the SYNTHIA dataset. Here, the images are not sourced from an existing consumer-grade game but from a custom virtual city environment in the Unity engine. In contrast to the above-mentioned approaches, the SYNTHIA dataset provides annotation data automatically as there is a higher-level access to the rendering process. Although the data generation is more flexible than using an existing game, it is still very focused on the autonomous driving context. A downside of this approach is also that the generation of the environment itself comes with a larger workload that is not present when using pre-existing game environments.

Another important point, discussed by Ros et al. [RSM+16] and also examined in greater detail by Tremblay et al. [TPA+18], Sun and Saenko [SS14] and Vazquez et al. [VLM+14] is the problem of domain shift. Domain shift describes the problem of training a computer vision algorithm on synthetic data and then deploying it in the domain of real data. When an algorithm is trained exclusively on synthetic data, it performs worse than an algorithm that was trained on real annotated data. A common solution to reduce the effects of domain shift is to use large amounts of synthetic data in combination with a small set of real annotated data. This approach led to results comparable to all-real training data in [RSM+16] and [VLM+14]. Another way of coping with domain shift used in object detection is training on a more abstract dataset altogether, eliminating false learning effects. This was demonstrated to be effective by Tremblay et al. [TPA+18] as well as Sun and Saenko [SS14].

All synthetic data generation approaches presented so far focus on the context of autonomous driving, which makes them lack the flexibility needed to provide sufficient data in the video surveillance and object tracking context of this thesis. Here, we need the ability to construct scenarios outside of the road-based autonomous driving context.

A different context where frameworks for generating synthetic ground-truth data are used is the understanding of indoor environments. SceneNet [HPB+16] is a repository of annotated synthetic indoor scenes that uses existing CAD models as a basis. These models from existing model repositories are arranged into indoor scenes, which are then rendered through virtual cameras at random viewpoints. For all rendered outputs, per-pixel labels are generated for all objects in the scene. A disadvantage of the automated scene generation from existing CAD models is that the generated outputs are static. McCormac et al. [MHLD17] improved on this by generating camera trajectories moving through SceneNet scenes. Here, the focus is more on creating photorealistic images using ray tracing, which leads to non-real-time outputs. Furthermore, the scenes themselves are still static with no object movement or deformations.

The most similar approach to our framework that can be found in previous work is the *ObjectVideo Virtual Video (OVVV)* video surveillance simulation. The video surveillance

testbed by Taylor et al. [TCB07] uses a modified version of the Half-Life 2 game by Valve Software to generate labeled video surveillance ground-truth data. Here, multiple virtual cameras are placed into the game environment, where character movements are recorded with their respective ground-truths as pixel labels and bounding boxes. Rendered outputs from the game engine are post-processed to include noise and ghosting artifacts. The resulting video is streamed to a recorder. In contrast to other game-based ground-truth generation frameworks, *OVVV* allows for more customization, as user-created maps can be loaded into the game via the Source Engine SDK. While it was one of the biggest inspirations for our framework, this project has since been abandoned by the developer and the sources are no longer available.

In contrast to previous work, such as the SYNTHIA dataset, which also relies on the Unity engine for rendering, our framework is not geared towards the context of autonomous driving but towards property-scale video surveillance. For this purpose, the framework focuses on tools that other synthetic data generation frameworks do not provide. One of them is the ability to remodel real walk patterns for the virtual characters using a waypoint system. Another new contribution of our framework is the ability to define critical areas that trigger an alarm in the case of certain character-environment interactions. This way the framework not only records visual data, but also data about events happening within the scene.

CHAPTER 3

Workflow and Implementation

The synthetic data generation workflow presented in this thesis consists of three main components. The first part consists of creating a model of scene as a basis for simulating scenarios in the framework. After that, the parameters of the framework in Unity get adjusted via the Scenario Editor plug-in. The final component is the rendering and data output which provides the end results of the data generation.

In this chapter we discuss the general workflow for generating synthetic video data with the framework. Furthermore, the most important implementation details of the Unity plug-in that comes with the framework are also presented here.

3.1 Overview of the Workflow

To get a better idea of the synthetic data generation process, we first take a look at the whole workflow from start to finish for one specific scene. The starting point in this case is a real-world scene, the side entrance to an office building, which is to be recreated in the framework. Since there is no preexisting 3D model of this particular scene, the first task is to create such a model. Based on photographs as well as measurements at the real scene, a to-scale model is built in Blender 3D [1]. This model contains all the structural elements that will be seen in the scene. Additional elements such as foliage, characters and surface textures are left out at this stage since they get added to the scene in the Unity editor.

Once the basic model is complete, it is imported into the Unity editor, where elements such as trees and brushes are added through external asset packs until the scene resembles its real-world counterpart. The finished scene can now be prepared for the simulation by creating a Unity NavMesh. The NavMesh defines all walkable areas in the scene via an

[1] `https://blender.org/` Accessed: 10.09.2019

auto-generated navigation mesh that is baked onto the environment. All characters that get added to the scene later on will use this mesh for their path-planning. A detailed description of the character navigation and path-planning can be found in Section 3.2. In the scene created for the evaluation, an Off-Mesh Link is also added to the NavMesh to bridge two areas that would otherwise not be traversable by the characters. The placement of this link is shown in Figure 3.1. Since the framework supports different time of day settings for the scene, additional artificial light sources are placed in the scene to provide illumination during nighttime settings.

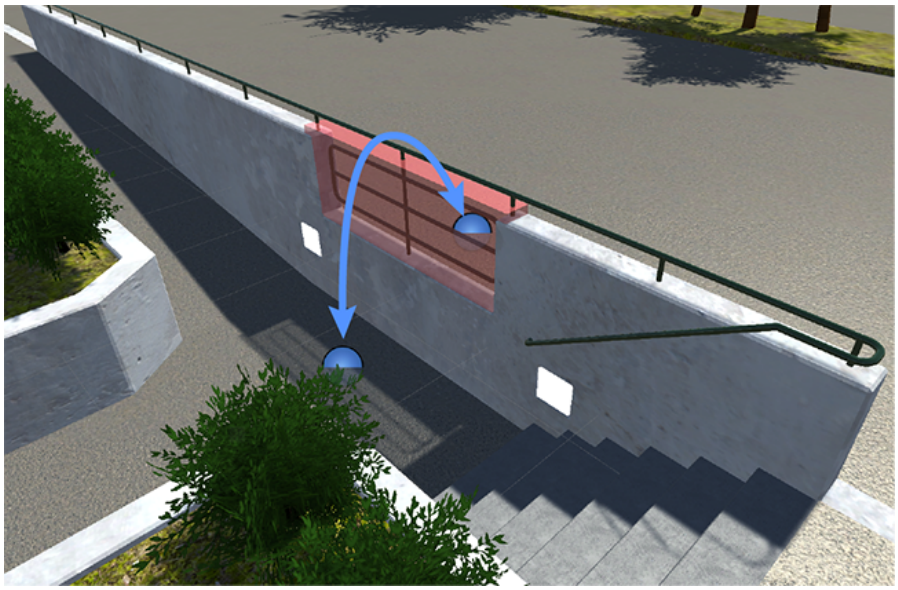


Figure 3.1: The Off-Mesh Link (indicated by the blue arrow) enables characters to jump over the fence between the two levels. The red volume marks a critical zone that will trigger an alarm when entered by a character climbing over the fence.

With the scene setup completed, the scenario for the recording is set up. In order for the characters to traverse the scene, a spawn point and one or more waypoints must be provided. These points are Unity Game Objects providing positional inputs for the character navigation. Additionally, critical zones can be defined in the scene. These zones will trigger an alarm when entered by a character. Figure 3.1 depicts such a zone where it is illegal for a character to climb over the fence. As the recorder utilizes Unity's cameras, camera Game Objects are placed in the desired locations from which the scenario should be recorded. In the case of the evaluation scene, the position and orientation of the virtual camera are matched to the camera in the real scene.

Before starting the final recording, the scene settings can be adjusted in a custom Unity editor provided by the framework. Alternatively, a random seed is used to generate a scenario. When the recording process is started, each frame generated by Unity is saved as an image two times. The first version contains the scene as rendered by the game engine, and the second version contains the label masks for each character in the scene.

Apart from the image sequence, the recorder also generates a JSON file that contains all the scene parameters as well as a list of characters, cameras and alarm events.

3.2 The Recording Framework

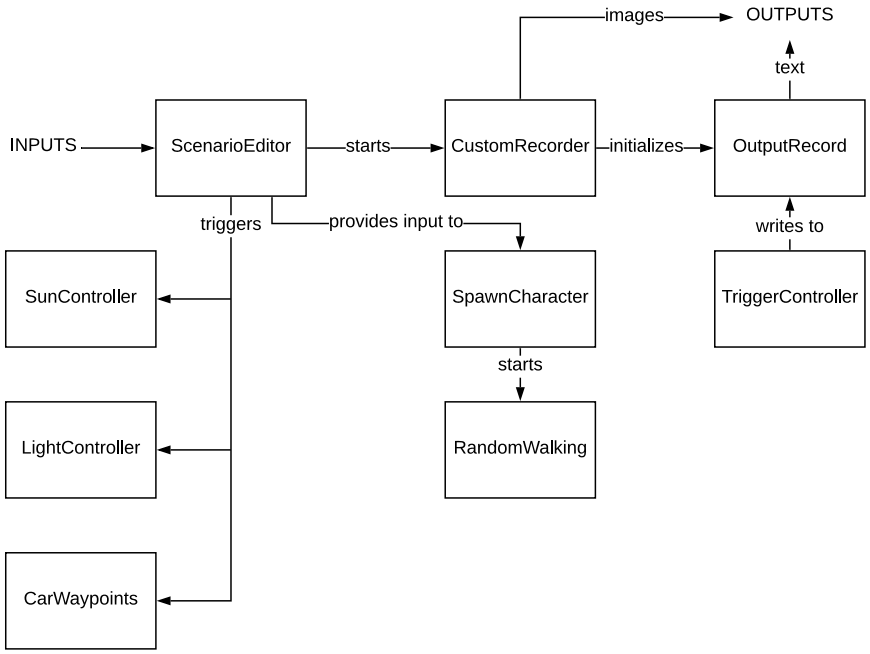


Figure 3.2: Schematic overview of the framework and its classes.

The central element of the framework for recording the synthetic videos is the Scenario Editor. This custom Unity editor is responsible for setting up the scene with the desired parameters as well as starting and stopping the recording process itself. Figure 3.2 depicts an overview of the recording framework. User input or a file with the desired parameters is given to the Scenario Editor. From there the settings are distributed to the according Game Objects such as lights and characters to setup the scene. When the recording is started, the Scenario Editor starts an instance of the Custom Recorder class, which is responsible for all graphical outputs, namely rendering and persisting the synthetic frames as well as the corresponding labels. In addition to its own outputs, the Custom Recorder also initializes an Output Record object that manages the textual output. Here, the scene parameters provided to the Scenario Editor as well as a list of critical events recorded by the Trigger Controller are saved in a JSON file.

Parameters for a scenario are provided in two ways: Environment settings such as the time of day, the wind speed and the number of characters as well as post-processing settings such as noise, blur and bloom are manually entered in the editor window or loaded via a JSON file. Other settings such as the waypoints for the character navigation

and the character spawn points are defined in the Unity editor. Adding new cameras is possible through the Scenario Editor as well. Since matching real camera views in Unity is one of the priorities of the framework, the camera view can be adjusted in the Unity viewport and the Scenario Editor is then used to instantiate a camera object with the parameters of the current viewport.

A camera object, which is called Camera Controller in the framework, consists of three Game Objects. The first one is the parent object of the cameras, which handles all transformations for the camera. Under this parent object there are two Unity camera objects, which are responsible for the scene recording. The Main Camera object renders the scene with the render settings from the game engine. All the post-processing scripts are attached to this object. The second child of the Camera Controller is the Label Camera, which is responsible for rendering the label information for all characters in the scene. For this, all the shaders in the scene are replaced by a black unlit shader and all objects with a label color property are rendered in this one solid label color. The rendering of the frames to image files is done by rendering the scene to a render texture and then saving the texture buffer to a PNG image. This process is done for every camera in all camera controllers of the scene.

The most important dynamic objects in the scene are the character models. To provide variation in the appearance of the characters, eight different human character assets from the Unity Asset Store are used in the framework. A character is controlled via the Unity Third Person Character script, which handles body movement of the rigged models. When a new character is spawned in the scene, a unique label color is assigned to the character and noted in the output log file. The chosen colors in any given scene are selected to have the greatest possible distance in hue, enabling the labels to be processed even in compressed video streams.

Character navigation is influenced by two factors: The NavMesh, which was generated when setting up the environment, provides data on the walkable areas and the user-defined waypoints, which provide destinations for the characters. All character navigation is controlled by the Random Walking class, which determines a destination and speed for the character. Navigation can happen in two ways: for completely automatic scene generation, the next waypoint is chosen at random, providing greater variation in character behavior. Alternatively, waypoints can also be traversed in a fixed order, as provided by the user. This second option is used for reconstructing real videos for the evaluation in Section 4.4. To enable more randomized paths for the characters, the destination determined in the navigation class is not set directly to the waypoint location but to a random point within a two-unit radius of the waypoint. Figure 3.3 depicts a possible path a character could take in the example scene. In the shown case the spawn point is at the entrance in the bottom of the frame, and the three waypoints are traversed in a clockwise circle starting from the leftmost point. Note that the actual destination deviates from the exact location of each waypoint. This way a second character traversing the same three waypoints would take a slightly different path.

Even more randomness in the character navigation is achieved by varying the cost of

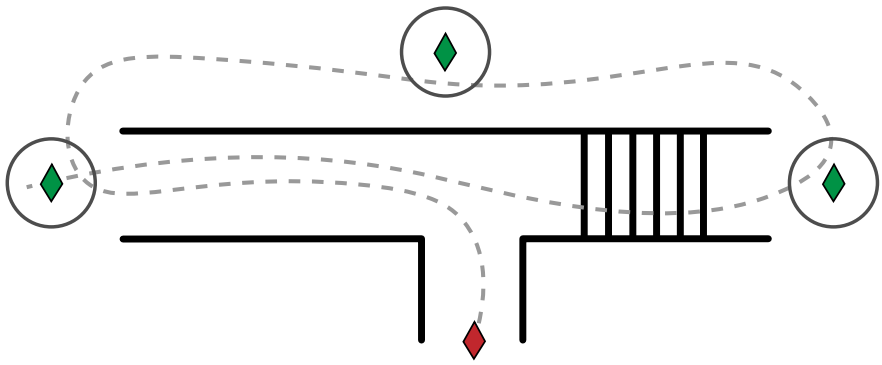


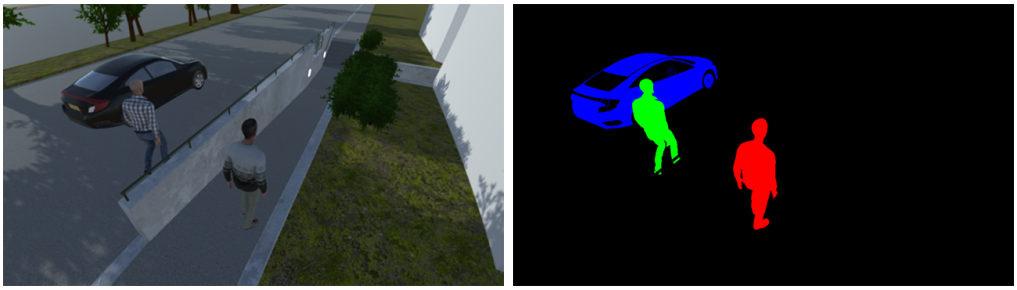
Figure 3.3: Possible path of a character, walking from the spawn point (red) to three different waypoints (green) in the sample environment.

Off-Mesh Links. This cost value associates a link element with a cost value that equates to a virtual walk distance. Since characters always choose the path of least distance, this cost variation influences whether an Off-Mesh Link shortcut is taken by a character.

The output data for a scenario consists of visual and textual outputs. The visual outputs are the rendered image frames from the camera objects. To build a consistent dataset, the video frame rate is fixed at 25 frames per second with a resolution of 1280 by 720 pixels. Each frame and its labels are recorded as a PNG file with lossless compression. In the label images, each character or object of interest is represented as a blob of solid color, where the color is the game object's label color. Figure 3.4 shows an example output from the framework, where multiple characters and a car are represented in the label data.

The JSON output file contains the recording name, which is the timestamp of the recording, as well as all the settings from the scenario editor. These settings are:

- The random seed
- The number of characters in the scene
- A flag that shows if there is a passing vehicle in the scene
- The time of day of the recording



(a) Generated video frame. (b) Labels for the frame.

Figure 3.4: Two characters and a passing car in an example from the dataset.

- The wind speed
- A flag that shows if the sun intensity changes due to simulated cloud shadows
- The noise intensity
- Three flags that indicate if noise, blur and bloom are activated
- The cost override value for Off-Mesh Links
- The position of the spawn point
- The number of frames in the recording

From these values, the basic parameters of a scenario can be recreated from a previous output. In addition to the scene parameters, three arrays are also part of the text output. The first one contains all alarm events that are caused by a character entering a trigger object. In the second one all characters in the scene with their respective spawn time and label colors are listed. The third array contains all camera controller objects and their position and orientation in the scene.

CHAPTER 4

Evaluation

Since it is the goal of this synthetic data generation framework to substitute real labeled video data with computer generated data, it is important to look at metrics which can be used to evaluate the feasibility of this substitution. This evaluation process is done by selecting fitting metrics for comparing tracking datasets to then look at how different tracking algorithms fare in processing the synthetic data in comparison to real video inputs. With the selected metrics, the synthetic data are compared to the according synthetic ground-truth, and the real video data with the corresponding manually acquired labels. For this comparison, scenarios from real videos are recreated in the synthetic data generation framework. After the initial evaluation, multiple post-processing effects as well as a variation in scene parameters of the synthetic videos are evaluated to see how such changes influence the performance of the tracking algorithms. In this second part of the evaluation, only the synthetic data are analyzed and compared to their respective ground-truth.

4.1 Evaluation Metrics for Tracking Algorithms

Finding an objective and comparable metric to evaluate the performance of a computer vision algorithm is a non-trivial problem. This section gives an overview of several established performance evaluation metrics for object tracking algorithms and their feasibility for evaluating tracking algorithms using our synthetic dataset.

Different approaches to performance evaluation of tracking algorithms have been discussed in previous work [NB03] [YMV07]. Since the focus of the tracking algorithms in this paper lies on generating trajectory data for the tracked objects, metrics for trajectory comparisons are the most obvious way to evaluate the data. The tracking algorithms used in this evaluation generally provide data output in the form of bounding boxes. Here the position of the bounding box centroid over time provides the trajectory data.

Apart from this trajectory data, the bounding boxes themselves can also be used for comparison.

Needham and Boyle [NB03] discuss metrics for comparing trajectories. A trajectory is defined as a sequence of positions over time. The trajectory T with the positions (x_i, y_i) at the times t_i is defined the following way:

$$T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n)\}. \tag{4.1}$$

Since applications using video footage as input data already have frames as equal time steps, the frame number (i.e. the index of the position) can be used as time step. In this case no additional timing information needs to be saved.

The simplest way of comparing two trajectories such as the tracker results and the ground-truth data is to calculate the Euclidean distance between the two trajectories at a given time step i. By doing this at every time step and then calculating the mean of the resulting values, the mean distance of the two trajectories can be obtained. This, of course, comes with the problem that outliers can easily skew the results and other metrics such as median distance may be better suited for some comparisons.

Another distance metric that is commonly used in the comparison of trajectories is the *Fréchet distance* [EM94]. This metric is defined as the minimum bottleneck-cost over all 1:1 mappings of the trajectories A and B. With the mapping $\mu : A \to B$ and the maximum distance in this mapping given as $\max_{a \in A} d(a, \mu(a))$, the Fréchet distance is:

$$F(A, B) = \min_{\mu} \max_{a \in A} d(a, \mu(a)) \tag{4.2}$$

Intuitively it can be explained as a person walking a dog on a leash, where both are moving on their independent trajectories. They are both only allowed to move forward, but they can vary their speeds. The Fréchet distance is the minimum length of the leash so that they both can traverse their trajectories. This metric is often more accurate than other distance metrics as it takes into account the order of points on a trajectory.

Yin et al. [YMV07] give a broader overview of tracking algorithm evaluation metrics, which do not solely focus on distance metrics. One of the simplest metrics that does not utilize the same data stream as the above-mentioned distance metrics is the bounding box overlap. This metric provides a value which represents the relative overlap of the output bounding box and the ground-truth bounding box. With A_g being the area of the ground-truth bounding box and A_o the area of the output bounding box the relative bounding box overlap A_{bb} is calculated as follows:

$$A_{bb} = \frac{A_g \cap A_o}{A_g \cup A_o} \tag{4.3}$$

In contrast to the distance measure between trajectories, the bounding box overlap metric also takes shape of the tracked object into account. Therefore, the accuracy of the object recognition can be analyzed to some extent.

Yin et al. [YMV07] also present several other metrics to describe the quality of a trajectory. The following two of them are of importance for our evaluation:

- *Track fragmentation:* A track fragmentation error occurs when the output track is not continuous where the ground-truth track is. The track fragmentation error rate is defined as the number of times a track is fragmented. Under optimal conditions, this error rate should be zero, which means that the track is one continuous segment.
- *System latency:* The time delay between the first occurrence of an an object in the scene and the time it starts to be tracked by the system. This latency has to be taken into account when comparing to ground-truth data, as the missing track segments might be interpreted as inaccurate tracking, when the actual cause is the system latency. The system latency LT is defined as the difference in frames between the first tracked frame of the output and the first frame of the ground-truth.

4.2 Tracking Algorithms used for the Evaluation

While the main motivation for creating the synthetic ground-truth dataset is the evaluation and improvement of a proprietary tracking algorithm, for the evaluation of the dataset itself another algorithm is employed as well. This is done to ensure that the dataset is not biased towards specific characteristics of the proprietary tracking algorithm. This section gives a brief overview on the design of the algorithms used for the evaluation.

The proprietary tracking algorithm of our industry partner is based on the algorithm discussed in [SFH+18]. It is not only a tracking algorithm but it is also able to separate background and foreground objects. Thereby it is able to detect objects moving against the background and track their movements as continuous trajectories. The detection and tracking are done in three steps: background subtraction, clustering and tracking. In the first step a background separation algorithm segments the image into blocks to determine areas of interest. The blocks are represented by local binary pattern feature vectors. A background learning model then determines the background-foreground status using the feature vectors. In the next step the foreground blocks are merged to clusters, which can then be tracked over time. In the final step the optimal cluster for each object is obtained using spatial, temporal and shape distances. This optimal cluster for each object is calculated at each timestamp, enabling a continuous tracking of a known object.

The second algorithm used for the evaluation is the MIL tracker implemented in OpenCV [1]. This algorithm uses online Multiple Instance Learning (MIL) to do object tracking in a video given an initial object location in the first frame of the video. This means that

[1] `https://opencv.org/` Accessed: 10.09.2019

the algorithm itself does not detect the object. The initial detection step must be done manually at the first occurrence of the object in the scene. The MIL tracker is a classifier that is trained at runtime. The initial bounding box supplied by the user is used as a positive example, while image patches outside of the bounding box are used as negative examples. Additionally, several image patches in the neighborhood of the initial position are also used to generate positive examples. All positive image patches are collected in a positive bag, which is then used by the MIL classifier for classifying the subsequent frames [BMB11].

Both algorithms provide the same type of output where a list of all tracked objects is generated for each frame. One data point of the output dataset contains the frame number, the object id of the tracked object and its bounding box. While our proprietary algorithm automatically manages multiple objects in the scene, the OpenCV MIL tracker has to be provided with an initial bounding box for each new object.

4.3 Evaluation Method

To evaluate the feasibility of substituting real video data with synthetic data, we use two different tracking algorithms to create trajectory data. We then compare the outputs from these algorithms to look for differences and possible shortcomings of the synthetic video data. The basis for this comparison consists of four video clips (referred to as `clip_1` to `clip_4`). These clips were recorded by a real surveillance camera on the premises of our industry partner, and for each of them, a ground-truth dataset was created manually. For each clip there is a corresponding reconstruction within the Unity framework, which allows for multiple simulations with the same basic parameters as the real video clip.

In the first step of the evaluation process, the real video clips are compared to their ground-truth data. This comparison provides a benchmark for the algorithm's performance on real data. In the next step, the synthetic reconstructions of the clips are compared to their respective ground-truth data. Finally, both results are compared to see whether the algorithms behave differently when analyzing synthetic data from our framework.

The comparison of the tracker output and the ground-truth trajectory is done using the metrics described in Section 4.1. In the first part, there is a quantitative analysis where the measures of mean and median distance, the Fréchet distance and the bounding box overlap are compared between each two trajectories. Secondly, we also take a look at track fragmentation and system latency as well as the overall performance of each of the two trackers. The results of this evaluation are discussed in Section 4.4.

Evaluation of trajectory similarity is done in R [2] using the package *trajectories* [PKM18], which provides methods for calculating the mean, median and Fréchet distances. For the bounding box overlap, the metric discussed in Section 4.1 is used to provide data for the comparison. Since the proprietary tracking algorithm does not provide output for

[2] `https://www.r-project.org/` Accessed: 10.09.2019

Clip	Length	Characters	Description
`clip_1`	302 Frames	1	Character walks through the lower section of the scene from right to left.
`clip_2`	354 Frames	2	2 characters walk from the lower exit to the left-hand border of the screen.
`clip_3`	310 Frames	2	2 characters walk from the left-hand border to the lower exit, car drives by afterwards.
`clip_4`	119 Frames	2	2 characters run from the left-hand border to the right-hand border while changing their heading multiple times.

Table 4.1: Overview of the clips used for the evaluation.

every frame in a given video, the ground-truth trajectory is resampled to contain the same number of frames as the tracking output for the bounding box overlap calculations. If this was not taken into account, the results for the overlap would be skewed by the fact that all dropped frames indicate a mismatch of 100 percent.

The four real video clips used for the evaluation are of a rather simple nature, with one or two people walking through the frame in a diffuse daylight setting. This reduced complexity helps to eliminate other disruptive factors such as weather changes. An overview of the parameters of the four clips can be seen in Table 4.1.

All four clips are set in daytime with diffuse lighting and good contrast. The clips start with the characters being out of frame and end the same way. There are no complex inter-character interactions such as occlusions or large deformations. The characters do, however, change their heading and speeds throughout the clips. Figure 4.1 gives an overview of the scene used for the evaluation. There are two types of trajectories in the evaluation, marked red and blue in the figure. Each of the trajectories is represented by two clips, where the two clips have the characters walk in opposite directions and at varying speeds.

The reconstructed synthetic clips have the same basic parameters as the real videos. These parameters are the length of the clip, the number of characters in the clip, the trajectory the characters traverse and the diffuse daylight setting. The timing of the reconstruction is matched to the original clip as closely as possible by tweaking the placement of waypoints and the spawn point in the framework. The virtual camera that is used to record the reconstructed scenes is modeled after the real camera by matching its optical parameters as well as its position in the scene. This matching is done by first setting up the measured parameters from the real camera and then also fine-tuning the exact transformation of the camera manually in Unity. For this fine-tuning, a canvas object with a frame from the real video is overlaid in the Unity camera view, allowing for visual matching.

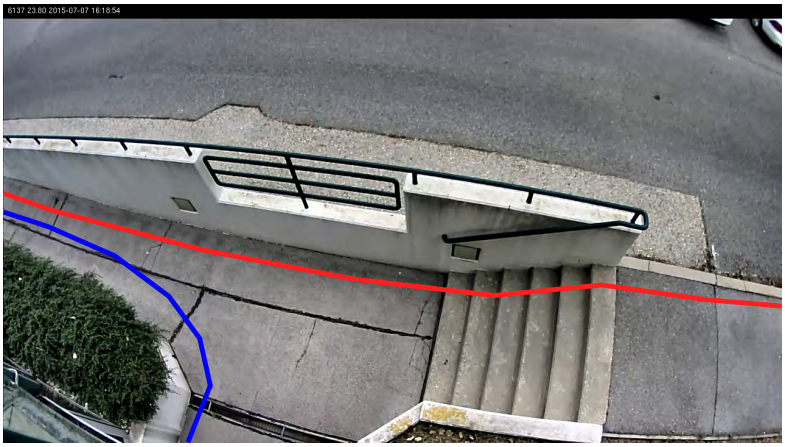


Figure 4.1: Overview of the real scene used for the evaluation. The red path corresponds to the trajectories in clips 1 and 4, while the blue path represents the trajectories in clips 2 and 3.

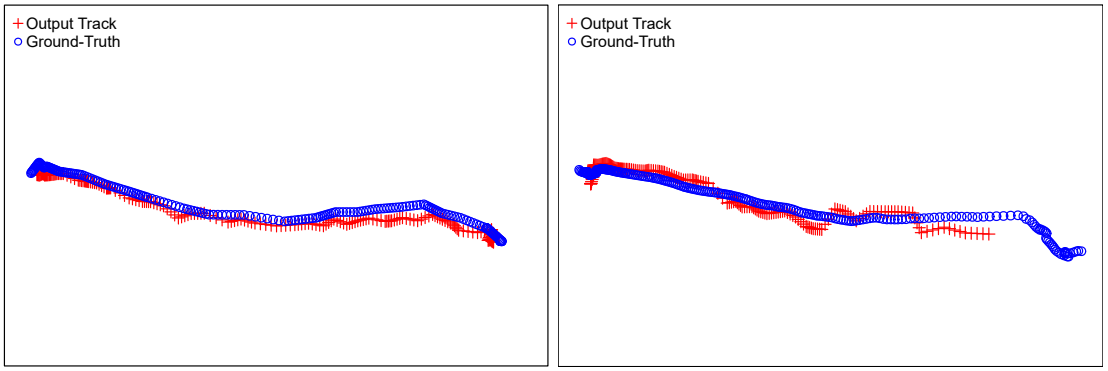
Clip	Tracker	BB Overlap	Mean Dist.	Median Dist.	Fréchet Dist.
Original	Prop.	0.7276	21.4784	21.0582	40.0575
Reconstructed	Prop.	0.5695	22.3377	21.7123	238.7950
Original	MIL	0.1159	97.6820	78.2086	174.6877
Reconstructed	MIL	0.0333	497.2692	524.8550	800.3849

Table 4.2: Tracking metrics for the proprietary and OpenCV MIL algorithms on both versions of clip_1.

Apart from the direct performance comparison of real and synthetic data, we also evaluate the influence of different parameters, such as time of day, the number of characters, noise and lighting changes on tracking performance. For this purpose, the same basic scenes from the video reconstructions are also rendered with completely different parameters and the tracking performance on these synthetic videos is then analyzed. The results of this secondary evaluation are discussed in Section 4.5.

4.4 Tracker Performance on Real and Synthetic Video

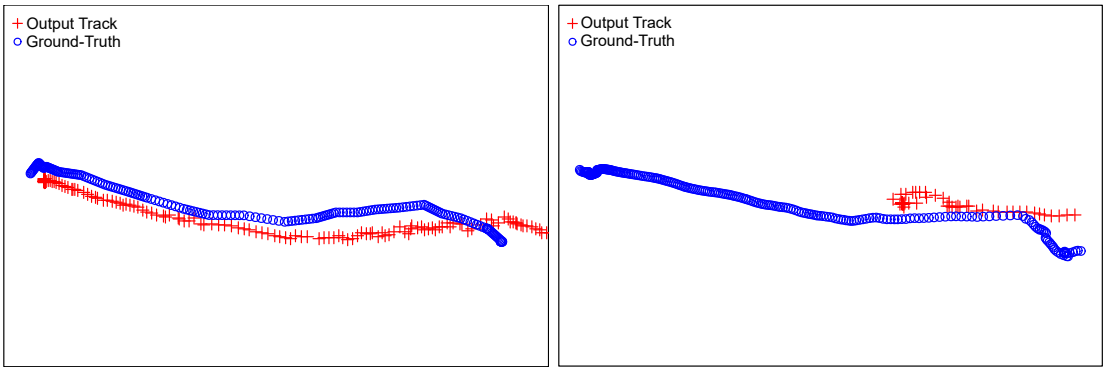
In the performance evaluation, we take a look at the results of the two tracking algorithms for each video clip. Overall, it is of notice that the proprietary tracking algorithm performs better than the OpenCV MIL tracker. However, since the goal of the evaluation is not to benchmark the tracking algorithms but to see if different trackers treat the synthetic data the same way as real inputs, this performance difference is not a problem. Note that all figures in this section do show the trajectories in their relative position to the whole camera frame.



(a) Original video. (b) Synthetic video.

Figure 4.2: Trajectory outputs of the proprietary tracker for clip_1.

The simplest video clip (`clip_1`) contains one person walking through the frame from right to left. The most challenging element of this clip is the area of high contrast by the railing of the stairs. The proprietary tracker performs very well on the original video clip, with a bounding box overlap of 72 percent and a Fréchet distance of 40 pixels. In the synthetic reconstruction of this clip, the metrics for the proprietary algorithm are significantly worse. Looking at the plots of the two trajectories and their ground-truth in Figure 4.2b, we can see that the main reason for this is tracker latency. The proprietary algorithm picks up the track of the character after about 60 frames. This latency affects the bounding box overlap and Fréchet distance, while both the mean and median distance as well as the graphs in Figure 4.2 show that the two output trajectories are similar.



(a) Original video. (b) Synthetic video.

Figure 4.3: Trajectory outputs of the OpenCV MIL tracker for clip_1.

With the OpenCV MIL tracker, the results diverge much more significantly. Here, the tracker loses the target character in the synthetic video at the high-contrast area around the stairs (see Figure 4.3b). In the real video clip, no such error occurs with the OpenCV MIL tracker. However, here there is a wider gap between the two trajectories (see Figure

Clip	Char	Tracker	BB Overlap	Mean Dist.	Median Dist.	Fréchet Dist.
Orig.	1	Prop.	0.6792	26.5907	25.0092	52.9008
Synth.	1	Prop.	0.3674	63.8594	56.1408	67.0422
Orig.	2	Prop.	0.7291	11.6062	11.3360	32.2437
Synth.	2	Prop.	0.6573	21.8230	17.0464	90.2694
Orig.	1	MIL	0.1560	42.2221	35.6078	51.2794
Synth.	1	MIL	0.1157	69.4341	69.0311	78.7873
Orig.	2	MIL	0.1259	45.8849	42.9530	57.1987
Synth.	2	MIL	0.1156	75.7855	79.2750	104.6802

Table 4.3: Tracking metrics for the proprietary and OpenCV MIL algorithms on both the original videos and the synthetic reconstruction of clip_2. Char 1 and 2 indicate the two different characters in this clip.

4.3a), which is most likely attributed to the fact that for the MIL tracker, larger bounding boxes were used. This larger bounding box then leads to a shift in centroid position, which in turn leads to the gap between the output tracks.

The second clip contains two characters walking from the bottom of the frame to the left. Since the two characters in `clip_2` do not interact or overlap, the clip does not provide a specific challenge to the tracking algorithms. Both characters walk in front of low contrast backgrounds, providing good visible separation from the background. The tracking metrics in Table 4.3 show that the overall performance in this clip is better than for `clip_1`. With the proprietary tracker, the results are once again slightly better for the original clips. The biggest difference between real and synthetic input data appears in the results for character 1, where the proprietary tracker output trajectory for the synthetic video is shifted to the left (see Figure 4.4). A similar shift can be observed in the tracking results for character 2 from the OpenCV MIL tracker. In general, the distance measures for the synthetic clips with the MIL tracker are significantly worse than for the real clips. There are no fragmentation or system latency-related problems for any of the versions of `clip_2`. All tracking attempts provide good trajectory outputs.

In the third clip, two characters traverse a similar path as in `clip_2` in the opposite direction. This clip has the best match in results for its real and synthetic versions over both tracking algorithms. The OpenCV MIL tracker once again has problems with tracking the characters to the end, but these problems are not exclusive to either the original or the reconstructed, synthetic video. Table 4.4 gives an overview of the trajectory metrics for this clip. Note that the Fréchet distance for the synthetic clip of character 1 from the proprietary tracker is an outlier. Here, a track fragmentation occurred, which led to the main trajectory being shorter than the ground-truth.

The fourth clip that is part of the evaluation has a trajectory similar to the one on `clip_1`. The main difference to the first clip is that there are two characters and they are both moving at a running speed in the opposite direction of the trajectory in `clip_1`.

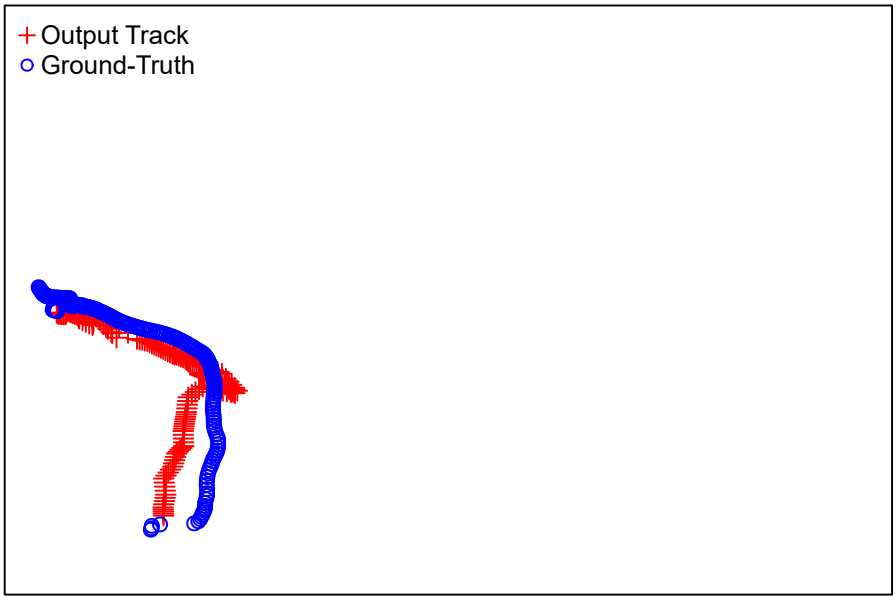


Figure 4.4: Trajectory output of the proprietary tracker for character 2 of the synthetic reconstruction of clip_2.

Clip	Char	Tracker	BB Overlap	Mean Dist.	Median Dist.	Fréchet Dist.
Orig.	1	Prop.	0.6602	21.2112	12.1562	79.8135
Synth.	1	Prop.	0.6180	34.2341	33.6684	152.0201
Orig.	2	Prop.	0.6903	16.0646	13.3678	69.6353
Synth.	2	Prop.	0.7201	19.5719	16.5491	53.4766
Orig.	1	MIL	0.1476	59.8323	54.6002	62.8073
Synth.	1	MIL	0.1401	85.0669	57.6581	147.6576
Orig.	2	MIL	0.1124	65.5799	65.1632	61.4593
Synth.	2	MIL	0.1251	72.0307	58.7154	149.3285

Table 4.4: Tracking metrics for the proprietary and OpenCV MIL algorithms on both the original videos and the synthetic reconstruction of clip_3. Char 1 and 2 indicate the two different characters in this clip.

In the real video, both trackers have no problems tracking both characters throughout the whole clip. The centroid position returned by the MIL tracker is shifted once again, resulting in sub-optimal performance metrics. For character 1, this shift is especially strong in the latter two thirds of its trajectory. This indicates that the bounding box shifted from the center of the character to following the boundary between the character and the background. The proprietary algorithm, on the other hand, performed excellently on the real video input. In the synthetic reconstruction, there were similar problems in the results for both algorithms. Character 2 causes track fragmentation errors in both

trackers, where only the first third of the trajectory can be conclusively reconstructed. Character 1 has similar problems in both trackers, albeit the fragmentation only happens in the last third of the trajectory.

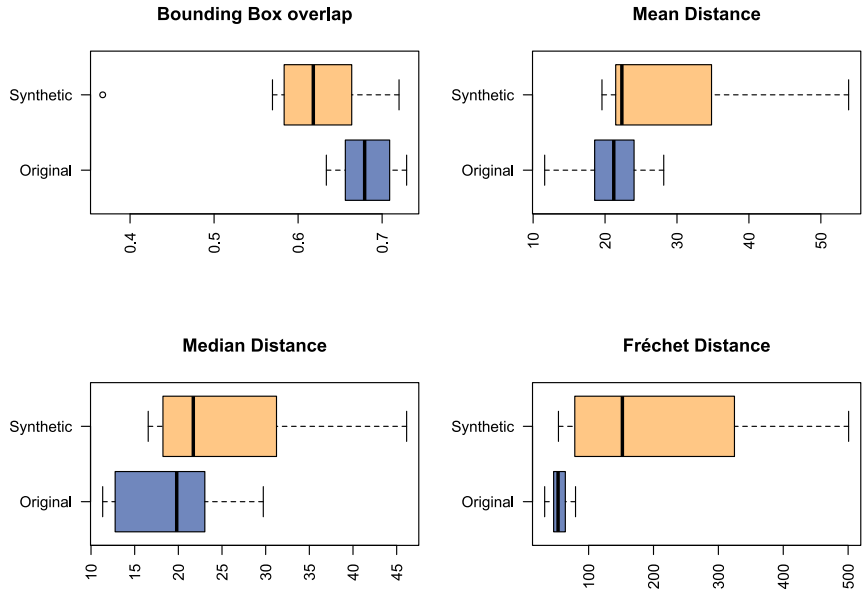


Figure 4.5: Overview of the proprietary algorithm tracking performance metrics for all four synthetic clips compared to the performance for the real clips.

We can see that both tracking algorithms perform better on the real video clips than on the synthetic input data. This may be attributed to the fact that these algorithms are optimized towards tracking objects in real videos. However, the difference in performance is not significant enough to conclude that synthetic data cannot be used to substitute real videos to test tracking scenarios.

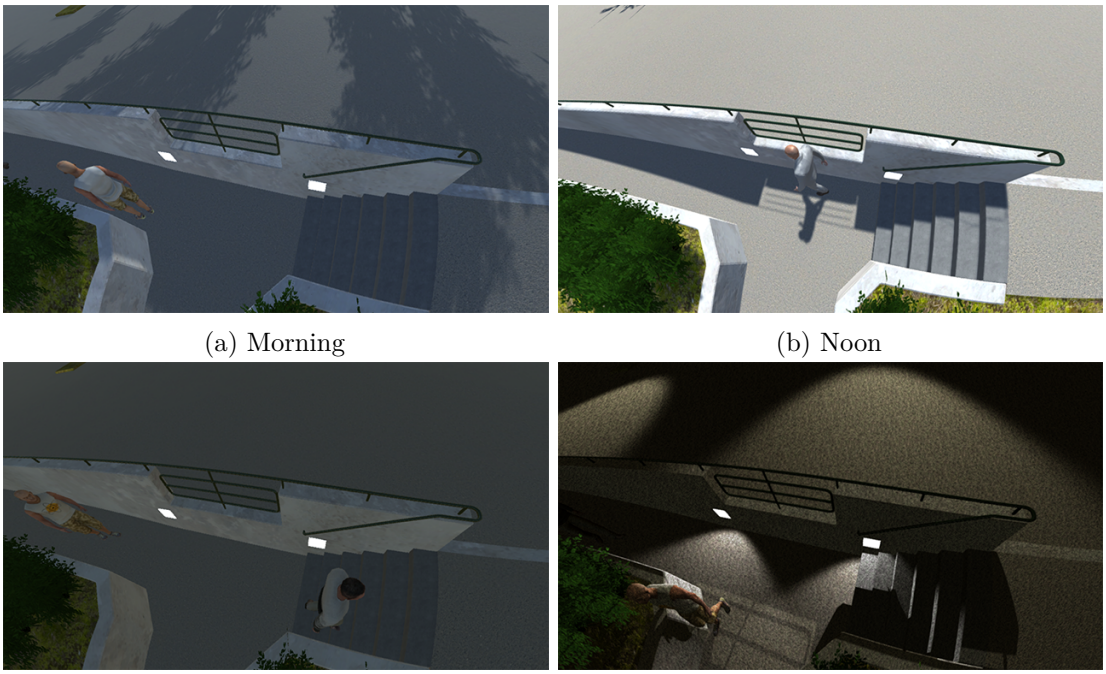
Figure 4.5 provides an overview of the tracking metrics for all synthetic and real video clips in the evaluation. Here the differences in the Fréchet distance are especially visible, where the trajectories obtained from the synthetic clips have a significantly higher variance, as well as higher distance values overall. One reason for this is that many of the trajectories from synthetic clips were shifted due to shadows of the characters being misidentified as part of the character. The real video clips were recorded in a more diffuse light setting, resulting in less prominent shadows. The other two distance metrics also show a behavior similar to the Fréchet distance, which is also mainly due to shifted trajectories. Except for one outlier in `clip_2`, the bounding box overlap for all clips stays above 55 percent.

This evaluation also only utilizes a very small sample size with four short clips for a total of nine different trajectories to be tracked. An evaluation with additional datapoints can be found in Section 4.5, where we take a look at how different post-processing effects

affect the tracking performance. Since the main goal of this thesis is to create a synthetic dataset, this evaluation should be seen as more of a proof-of-concept than a definitive statement that synthetic data should or should not be used in place of real training and test data.

4.5 Influence of Post-Processing and Parameter Variation

While the scenario parameters for the clips were kept the same for the initial evaluation, the framework is also capable of producing different settings. In this section, we evaluate whether built-in Unity effects are effective in producing a more versatile dataset. For this evaluation, the reconstructed synthetic video clips from Section 4.4 are rendered with variations in the time of day, the character model and different post-processing effects. An overview of the visual impacts of the time of day variations can be seen in Figure 4.6.



(a) Morning (b) Noon

(c) Evening (d) Night with noise

Figure 4.6: Visual impact of the different time of day settings in the synthetic video clips.

For the time of day, four options are available, where each one comes with specific challenges:

- *Morning:* There is a high contrast between illuminated and not illuminated areas, as well as long shadows producing additional visual clutter in the background.
- *Noon:* This is the standard setting, which is also used for the main evaluation. Here, the scene is evenly lit, and shadows are short.

- *Evening:* At this time of day setting, there is no direct sunlight on the scene, resulting in a diffuse illumination.
- *Night:* Here, no parallel light source is available, and the scene is only lit by the artificial light sources placed in the scene. This setting is especially challenging, since there are areas with very low contrast.

The three post-processing effects in the framework each correspond to effects appearing in real-life video surveillance scenarios:

- *Noise:* Noise and grain appear in low-light scenarios and can lower the contrast or impede clear image segmentation.
- *Blur:* Blurred images can be the result of dirt or moisture on a camera lens.
- *Bloom:* Bloom happens when extremely bright areas of an image start to bleed into neighboring areas, which can lead to potential problems in image segmentation.

Apart from the above-mentioned parameters and effects, the wind speed is also variable. A higher wind speed results in fuzzy movements of the vegetation model, which in turn creates moving shadows on the scene's surface. These shadows can also potentially lead to false positives in tracking.

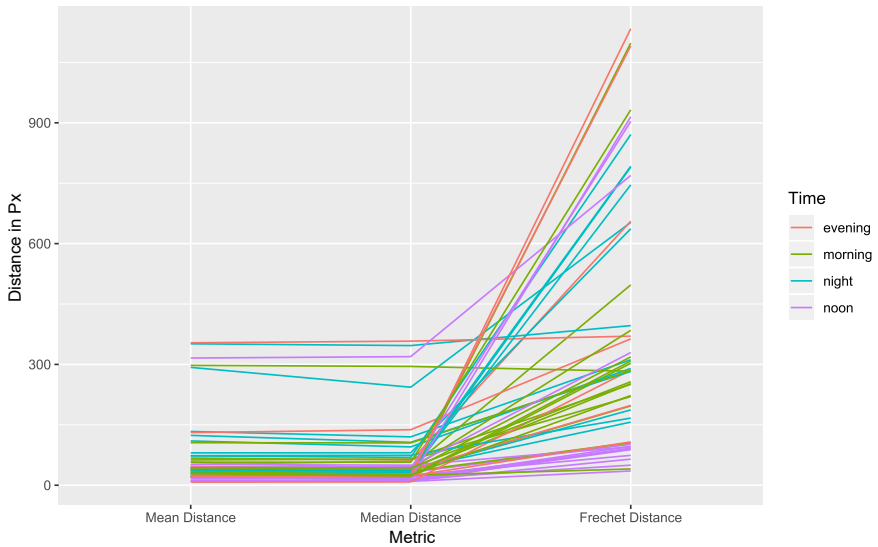


Figure 4.7: Distance metrics for all clips tracked with parameter variation using the proprietary tracking algorithm from our industry partner. Each line represents one clip, color coded by the clip's time of day.

As shown in Figure 4.7, the different parameters do have significant influence on the quality of the tracking results. The main observation is that the night setting is most

likely too dark and lacks contrast, leading to an overall higher tracking error for all metrics. Another issue seems to be high noise values, as all the outliers from other time settings with larger errors are clips with high noise values. To facilitate more usable results, the noise value in the framework should therefore be kept low.

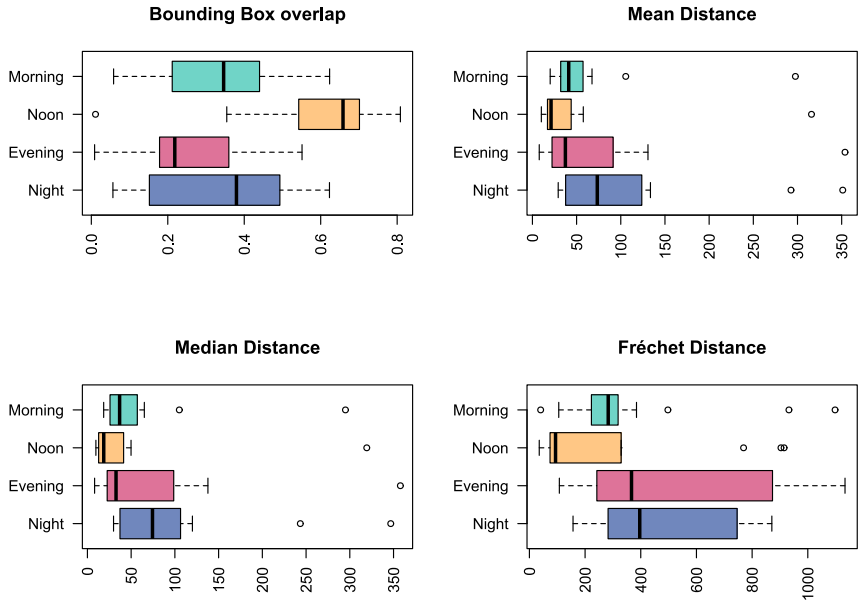


Figure 4.8: Overview of the proprietary algorithm's tracking performance metrics for the four different time of day settings. Note that additional post-processing effects are applied to some of the clips, leading to a greater variance in each time of day setting.

The graphs in Figure 4.8 illustrate another problem, which is the extremely low contrast in the evening setting. Here, even without additional noise, the tracking performance in some cases is even worse than in the night setting. The low contrast in the evening setting can also clearly be seen when comparing the sample frames in Figure 4.6c and 4.6d. Another observation that can be made by comparing the results for the noon setting in Figure 4.8 with the results from the synthetic data in Figure 4.5 is that the tracking performance with post-processing effects is mostly comparable with the unaltered output. The only difference here is once again in the clips with excessive noise, which are responsible for the outliers in the plots.

Overall, this quick evaluation shows that the built-in Unity post-processing options provide a good basis for introducing variety to the synthetic dataset. However, more fine tuning is still needed to find a balance between variation and acceptable tracking results.

CHAPTER 5

Conclusion

We have shown that generating synthetic ground-truth datasets is possible within the Unity game engine. Except for the manual task of content creation, the presented data generation framework can be used for fully automated generation of video surveillance data. As an example for the output generated by our framework, the settings of a real video surveillance scenario were recreated within the game engine. From this scene, 64 synthetic video clips with varying parameters were created, where each clip was recorded from two different camera perspectives. This data generation resulted in 86,768 synthetic frames and their respective labels. The label data as well as the synthetic video frames themselves were generated with a real time render engine within our framework and recorded as single images by rendering to a texture and saving the buffer contents as an image. On the majority of the clips, additional post-processing effects from the Unity post-processing stack were also applied to introduce variation into the video clips. The semantic labeling was done automatically by assigning a unique label color to each object of interest and then re-rendering each frame with all shaders being replaced by the solid label colors.

About half of the synthetic clips were recreations of real video clips recorded from a virtual camera that matched the settings of a real camera from the scene. These clips were then used as input for two tracking algorithms to extract trajectory data from the characters traversing the scene. By comparing the resulting trajectories to the ground-truth data and doing the same for real, manually labeled clips, it was shown that the tracking algorithms perform worse on the synthetic data than on the real video data. This indicates that the synthetic video data in its current form is limited in its suitability for evaluating tracker performance. As none of the tested algorithms has been trained with the synthetic dataset, there were no definitive observations made as to how well the dataset is suited for the purpose of training. The effect of algorithms performing worse on synthetic videos may be similar to the known problem of domain shift. Here, the problem is that algorithms trained on synthetic data struggle when they then have

to analyze real inputs. One conclusion to be drawn from this is that the evaluation of tracking algorithm performance should not solely rely on synthetic data but at best utilize a combination of real and synthetic videos. Since the main aim of this thesis was to create the data generation framework itself, the evaluation was also only done on a rather small dataset. More conclusive results could possibly be obtained by comparing a larger dataset of both real and synthetic videos.

Overall, it was shown that Unity can be used as a tool to create synthetic video data in real time without requiring major modifications. Having full access to the source code and all resources provides an advantage over relying on off-the-shelf games when it comes to labeling, as no manual work is required.

5.1 Future Work

An important follow-up to the work presented here would most definitely be to train a tracking algorithm with the dataset generated by the framework. In this regard, the dataset will be used in the scope of the ALOHA project [1] to train deep learning models for behavior analysis. This could also provide further insight into the feasibility of using our synthetic training data and show how severe the differences between the real and our synthetic data are when it comes to a deep learning approach.

Even without the step of training a tracking algorithm with the generated data, the evaluation presented in this thesis could be improved by evaluating a larger, more comprehensive dataset and comparing it with more real annotated videos from different scenarios and environments.

Regarding the framework itself, multiple future extensions and improvements are possible. One of them could be making the scenarios more versatile by adding additional actors such as animals and different vehicles, more weather patterns such as snow, rain or dead leaves moving in the wind. More realistic character interactions with the environment or characters carrying objects could also be worth adding to the framework. The value of the generated synthetic data will likely increase with the ability to simulate more diverse scenarios.

With the need to create more diverse scenarios comes the problem of content creation. This problem already exists in the current state of the framework, where the 3D scene of the environment has to be created manually, but it will only become a bigger problem with the need of adding more complexity to the scenario. Possible future improvements in this area could be twofold. On the one hand, a more modular approach could be taken by creating more abstract scenes from simple, pre-made building blocks. On the other hand, techniques like photogrammetry could be investigated as tools for re-creating real environments for the framework. In general, a more automated and procedural workflow would definitely be a welcome improvement.

[1] `https://www.aloha-h2020.eu/` Accessed: 09.10.2019

When it comes to the visual output of the framework, a possible area for expansion could be to introduce more sophisticated post-processing to create more diverse image sets. Here it might be worth to look into style transfer methods such as CycleGAN where, in one example, a neural network was used to turn scenes from GTA V into more realistic video frames [LLJX18]. A similar approach could also be taken to generated different camera modes or seasons out of a single video recording.

List of Figures

List of Tables

Bibliography

[AEK+18] Matt Angus, Mohamed ElBalkini, Samin Khan, Ali Harakeh, Oles Andrienko, Cody Reading, Steven Waslander, and Krzysztof Czarnecki. Unlimited Road-scene Synthetic Annotation (URSA) Dataset. *arXiv:1807.06056 [cs]*, July 2018. arXiv: 1807.06056.

[BMB11] B. Babenko, Ming-Hsuan Yang, and S. Belongie. Robust Object Tracking with Online Multiple Instance Learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(8):1619–1632, August 2011.

[EM94] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

[HPB+16] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding RealWorld Indoor Scenes with Synthetic Data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4077–4085, Las Vegas, NV, USA, June 2016. IEEE.

[LLJX18] Peilun Li, Xiaodan Liang, Daoyuan Jia, and Eric P. Xing. Semantic-aware Grad-GAN for Virtual-to-Real Urban Scene Adaption. *arXiv:1801.01726 [cs]*, January 2018. arXiv: 1801.01726.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[MHLD17] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2678–2687, 2017.

[NB03] Chris J. Needham and Roger D. Boyle. Performance Evaluation Metrics and Statistics for Positional Tracker Evaluation. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, James L. Crowley, Justus H. Piater, Markus Vincze, and Lucas Paletta, editors, *Computer Vision Systems*, volume 2626, pages 278–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[PKM18] Edzer Pebesma, Benedikt Klus, and Mehdi Moradi. *trajectories: Classes and Methods for Trajectory Data*, 2018. R package version 0.2-1.

[RSM+16] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243, Las Vegas, NV, USA, June 2016. IEEE.

[RVRK16] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for Data: Ground Truth from Computer Games. *arXiv:1608.02192 [cs]*, August 2016. arXiv: 1608.02192.

[SFH+18] Natalia Shepeleva, Lukas Fischer, Thomas Hoch, Werner Kloihofer, and Bernhard Moser. Removing nuisance in tracklet data. In Henri Bouma, Robert J. Stokes, Yitzhak Yitzhaky, and Radhakrishna Prabhu, editors, *Counterterrorism, Crime Fighting, Forensics, and Surveillance Technologies II*, page 28, Berlin, Germany, October 2018. SPIE.

[SS14] Baochen Sun and Kate Saenko. From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains. In *Proceedings of the British Machine Vision Conference 2014*, pages 82.1–82.12, Nottingham, 2014. British Machine Vision Association.

[TCB07] Geoffrey R. Taylor, Andrew J. Chosak, and Paul C. Brewer. OVVV: Using Virtual Worlds to Design and Evaluate Surveillance Systems. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, MN, USA, June 2007. IEEE.

[TPA+18] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1082–10828, Salt Lake City, UT, June 2018. IEEE.

[VLM+14] David Vazquez, Antonio M. Lopez, Javier Marin, Daniel Ponsa, and David Geronimo. Virtual and Real World Adaptation for Pedestrian Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(4):797–809, April 2014.

[YMV07] Fei Yin, Dimitrios Makris, and Sergio A Velastin. Performance evaluation of object tracking algorithms. In *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, Rio De Janeiro, Brazil*, page 25. Citeseer, 2007.