

AUTOMATIC DETERMINATION OF NAVIGABLE AREAS, PEDESTRIAN DETECTION, AND AUGMENTATION OF VIRTUAL AGENTS IN REAL CROWD VIDEOS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Yalın Doğan
December 2018

Automatic Determination of Navigable Areas, Pedestrian Detection,
and Augmentation of Virtual Agents in Real Crowd Videos

By Yalın Doğan

December 2018

We certify that we have read this thesis and that in our opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Uğur Gündükbay(Advisor)

Hamdi Dibekliolu

Ramazan Gökberk Cinbiş

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

AUTOMATIC DETERMINATION OF NAVIGABLE AREAS, PEDESTRIAN DETECTION, AND AUGMENTATION OF VIRTUAL AGENTS IN REAL CROWD VIDEOS

Yalım Doğan

M.S. in Computer Engineering

Advisor: Uğur Güdükbay

December 2018

Crowd simulations imitate the behavior of crowds and individual agents in the crowd with personality and appearance, which determines the overall model of a multi-agent system. In such studies, the models are often compared with real-life scenarios for assessment. Yet apart from side-by-side comparison and trajectory analysis, there are no practical, out-of-the-box tools to test how a given arbitrary model simulate the scenario that takes place in the real world. We propose a framework for augmenting virtual agents in real-life crowd videos. The framework locates the navigable areas on the ground plane using the automatically-extracted detection data of the pedestrians in the crowd video. Then it places the three-dimensional (3D) models of real pedestrians in the 3D model of the scene. An interactive user interface is provided for users to add and control virtual agents, which are simulated together with detected real pedestrians using collision avoidance algorithms.

Keywords: Image processing, pedestrian detection and tracking, computer vision, three-dimensional reconstruction, computer graphics, crowd simulation, augmented reality.

ÖZET

GERÇEK KALABALIK VİDEOLARINDA GEZİLEBİLİR ALANLARIN BELİRLENMESİ, YAYALARIN TESPİTİ VE SANAL BİREYLER EKLENMESİ

Yalım Doğan

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Danışmanı: Uğur Güdükbay

Aralık 2018

Kalabalık benzetimleri, kalabalıkların ve içerdikleri ajanların bireysel davranışlarını, kişilik ve görünüşlerini örnek alarak, bir çoklu-ajan sisteminin genel modelini tanımlar. Bu tür çalışmalarda, modeller değerlendirilirken genellikle gerçek hayattaki senaryolarla karşılaştırılır. Ancak yan yana karşılaştırma ve güzergah analizleri dışında, verilen herhangi bir modelin gerçek bir senaryonun ne derece başarılı şekilde benzetimini yaptığını doğrulayan, pratik ve doğrudan kullanılabilir araçlar bulunmamaktadır. Bu çalışmada, yapay ajanları gerçek kalabalık videolarına eklemek için bir sistem öneriyoruz. Önerilen sistem ilk olarak kalabalık videosundan otomatik olarak elde ettiği yaya tespit bilgilerini kullanarak, yer yüzeyindeki gezilebilir alanları bulmaktadır. Sonrasında gerçek yayaların üç-boyutlu (3B) modellerini 3B sahneye yerleştirmektedir. Tespit edilen gerçek yayalarla çarpışma önleme algoritmaları kullanılarak beraber benzetimi yapılan yapay ajanlar, kullanıcılara sunulan etkileşimli bir kullanıcı arayüzü aracılığıyla eklenip, kontrol edilebilmektedir.

Anahtar sözcükler: Görüntü işleme, yaya tanıma ve takip, bilgisayar görüşü, üç boyutlu yeniden yapılandırma, bilgisayar grafikleri, kalabalık simülasyonu, artırılmış gerçeklik.

Acknowledgement

I would like to thank my thesis advisor Prof. Dr. Uğur Gdkbay for his support on all parts of my thesis and guiding me into computer graphics and crowd simulations. I express my gratitude to my colleague Serkan Demirci from Bilkent University for his comments and support, especially for the pedestrian detection part. Additionally, I would like to thank Dr. Hamdi Dbeklioęlu for his comments on computer vision and writing of this thesis. I would like to thank Ateş Akaydın for providing videos and models to be used in the simulation.

I want to thank my dear Glser, my mother Nurcan, my father Oktay, my grandparents and family as a whole for their encouragement, which made my education and this work possible. Lastly, I want to dedicate this thesis to my late grandmother Nazmiye, who always had her trust in me.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	4
1.3	Outline	5
2	Background and Related Work	6
2.1	Pedestrian Detection and Tracking	6
2.2	Data-Driven Crowd Simulations	8
2.3	Area Reconstruction	9
2.4	Augmented Crowd Simulations	12
3	Framework Overview	13
4	Pedestrian Detection and Tracking	16
4.1	Stabilization of the Video	17

4.2	Pedestrian Detection using HOG	18
4.3	Pedestrian Detection using Neural Networks	20
4.3.1	<i>Inception</i> Network	21
4.3.2	<i>MobileNet</i> Network(s)	22
4.4	Pedestrian Tracking	23
4.5	Posture Detection	25
4.5.1	Results	26
5	Image Segmentation for Navigable Area Extraction	32
5.1	Meanshift	33
5.2	DBSCAN	34
5.3	HDBSCAN	35
5.4	Results and Discussion	36
6	Navigation Area Reconstruction	41
6.1	Horizon Detection	43
6.2	Perspective Correction	46
6.3	Camera Placement	50
6.4	Navigation Mesh Construction	52
6.5	Results in Unity	55

<i>CONTENTS</i>	viii
7 Crowd Simulation and User Interface	61
8 Results and Discussion	65
Bibliography	76
Appendices	87
A Pedestrian Tracking Results	87

List of Figures

1.1	Example of augmenting agents into a video from Game of Thrones (©HBO Entertainment, 2015) [1]	3
3.1	The proposed framework.	15
4.1	The importance of stabilization for background subtraction	19
4.2	The simple visualization of SSD [3] layer structure using VGG- 16 [2] as base network.	21
4.3	The inception module, as described in [4].	22
4.4	Using Kalman filters between consecutive frames, we are able to predict the next location of the tracked pedestrian and correct it according to detection in the next frame.	23
4.5	The state diagram of the tracker.	25
4.6	Example postures from different videos	27
4.7	Example pedestrians postures from PETS video.	31
5.1	The resulting MST (a) and hierarchical structure (b) of a 20×20 gray scale image	38

5.2	Shadow detection and removal process on an example frame . . .	39
6.1	The overview of the navigable area reconstruction.	42
6.2	Difference in vanishing lines for different camera orientations . . .	43
6.3	From the tracking data, the head and foot positions of the pedestrian at different times create parallel lines in the real world, which defines a single vanishing point	45
6.4	Example perspective correction of Central Park in New York, USA	47
6.5	The overview of the metric rectification process for PETS09-S2L1	49
6.6	The model adjustment process	53
6.7	An example triangulation of a black and white image	55
6.8	Example triangulation of the navigable area.	56
6.9	Example horizons calculated using the line features and pedestrian posture trajectories in the image	57
6.10	Each video together with their perspective corrected versions . . .	58
6.11	The placement of navigable area model for each video	59
6.12	Differences in postures of dummy models with pedestrians in the video	60
7.1	The graphical user interface of the framework.	64
8.1	Screenshots from a simulation on PETS video.	67

8.2	Sample interaction between an artificial agent and a detection in video.	68
8.3	Sample detection and projection results for Video-1, Video-2, and PETS, from top to bottom, respectively.	69
8.4	Sample projections from Video-1.	70
8.5	A selected agent avoids collision with a projected agent and adjusts its path before preparing to avoid a couple in Video-1.	71
8.6	The agents in Video-1 drags an artificial agent in (b), which causes it to be left behind in (c) and (d).	72
8.7	Sample screen shots from Video-2.	73
8.8	A selected artificial agent avoids a couple of projected agents in Video-2.	74
8.9	The projected agents enforce the groups of artificial agents to form a line for collision avoidance (Video-2).	75
A.1	Further example pedestrians postures from PETS video.	88
A.2	Example pedestrians postures from Video-1.	89
A.3	Example pedestrian postures from Video-2.	90

List of Tables

4.1	The quantitative results of our pedestrian tracker	30
5.1	The quantitative comparison of tested segmentation algorithms. .	38
5.2	Segmentation results including pedestrian trajectories, segments and navigable areas. Colors are only for visualization, does not always indicate identical clusters.	40
6.1	The focal length values for test videos compared to the ground truth calculated from the annotated image cues.	56

Chapter 1

Introduction

1.1 Motivation

Crowd simulations is the study of investigating and simulating the behavior, animation, appearance and even emotions of individuals who compose it and their effect on the crowd as a whole, which is called crowd's model. Such crowd models are used to create a living environment where people in the crowd interact with their surroundings in a believable manner, which plays an important part in the entertainment industry; especially video games. In such simulations, agents' appearance and behavior need to fit the context of the scene. The quality of augmentation in all related aspects is crucial, as the accuracy of both appearance, behavior and animations have a huge impact on overall immersion.

Apart from the entertainment industry, crowd simulations are also used in civil engineering and defense industry in order to investigate the behavior of sparse/dense crowds under “serious” cases that cannot be easily recreated in the real world. For example, evacuation scenarios using Multi-Agent Systems (MAS) [5] needs to be simulated in order to assess the efficiency of designed evacuation process in the defined structural environment or a riot scenario to plan an intervention for. The realism plays an even more important role in such

simulations, as how accurate they reflect a real-life event may determine the survival of related personnel and people.

When it comes to assessing the realism of such simulations, researchers rely on several methods for capturing the underlying model of the real world data and its comparison with its simulation. One of the methods for obtaining data is to place sensors for motion detection in a predefined area in order to extract the pedestrian flow information [6]. Rather than using sensors, computer vision techniques related to pedestrian detection and tracking are also used to determine the behavior of pedestrians, which is partially practiced in this work. The obtained data is used to compare with the simulation in a quantitative way. Simulations that use such extracted information directly are called data-driven and more examples are going to be given in the following chapters.

In cases where quantitative data is not available or perception of real humans needs to be addressed, we require user evaluations; where a group of real humans assesses the underlying properties of crowd model in the sense of realism. In such evaluations, users are provided with visuals for side-by-side comparison. Users can also be asked to create a simulation in order to assess its usability [7]. User studies for evaluating the “blending” aspect of artificial crowds and studies on investigating, simulating the interactions between real and artificial agents require augmenting virtual agents in images or video sequences.

Methods for augmentation of objects are frequently used in the entertainment industry. Especially movies use such techniques as part of their visual effects (VFX). Figure 1.1 shows an example of the creation of an artificial crowd. Following a similar approach, the studied crowd models can also be augmented on such real-life scenes. Yet this method is not preferred because it introduces a lot of extra work to an already lengthy period; as modeling large scale and complex cases where more than a dozen of agents involved can take considerable time, especially modeling the navigable area to be used. In most of the scenarios, such areas are created manually where they are either too simple that they don’t accurately reflect a real-life case or take too much time when they are expected to match a real-life area in an acceptable level, which is not easily scalable. The

accuracy of such areas also plays an important role, as having agent walking on an area that is not supposed to be navigable would degrade the realism.

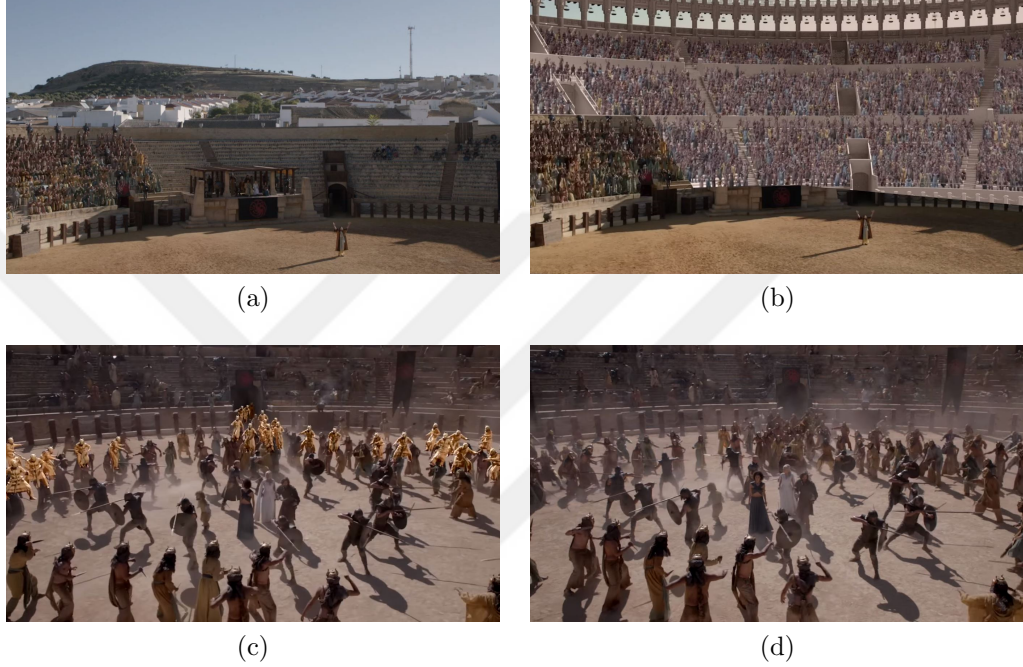


Figure 1.1: Example of augmenting agents into a video from Game of Thrones (©HBO Entertainment, 2015) [1]. The crowd of cheering actors in (a) are replicated and placed on reconstructed areas that reassembles a fighting pit in (b). In (c), a group of agents can be seen in gold color that looks almost identical to real actors in (d) after some post-processing.

We propose a data-driven framework for simulating artificial agents together with real-life pedestrians. The users can utilize this framework to assess the realism of their crowd models in a real-life scenario or expand upon it, by just providing its video footage; preferably from a surveillance camera with high altitude. The framework provides an automatic process that requires minimal manual intervention, which is related to camera calibration parameters.

The framework consists of two main parts: the first part is preprocessing the given video feed where the second part is an interactive crowd simulation. Preprocessing part includes the detection and projection of real pedestrians and determination of navigable areas used by the pedestrians. Real-life pedestrians in the video area located using pedestrian detection and tracking techniques.

Then the postures and trajectories of the pedestrians, combined with cues from the scene, are used for reconstruction of the navigable area. This way, users are saved from spending time on manually designing the area that is determined navigable by the people in the video. In the simulation part, a user interface is provided for adding and simulating artificial agents on the previously generated navigable area, where they blend with the real-life agents using collision avoidance algorithms. The user can also edit the placement of the navigable area, in order to adjust it according to the given video.

1.2 Contributions

The main contributions of the thesis are as follows:

- We developed a framework on simulating crowds over arbitrary videos where a sparse/dense crowd exists, which introduces scalability to blending scenarios.
- Framework can be directly used to assess the pedestrian behavior models in real-world environments with minimal effort in construction. The users can simulate the virtual agents beside real people to see if the simulated agents seem visually and behaviorally “realistic” enough.
- Works in cases where scene cues aren’t enough for reconstruction of the ground plane, where scene lacks man-made structures that contain features which their real-world counterpart is known: for example if they are parallel or not. Our framework uses pedestrian flow information to solve this problem of reconstruction.
- Navigable areas extracted according to the behavior of real pedestrians which is done automatically. We have also investigated certain segmentation methods for extracting those areas.
- Providing an extensive user interface for simulation of pedestrian behavior in the sense of collision avoidance and adjustable mesh placement for

increased usability.

- Our framework can be used to extend the scenes in given videos by adding more agents with a specific behavior, which can be used as data for further research. The framework can also be used to recreate the video in 3D automatically in order to save time.
- Our framework has multiple subsystems where they can be replaced with different techniques by the users. This makes our framework expandable and scalable to match the users' needs.

1.3 Outline

The thesis is organized as follows. Chapter 2 gives background information on crowd simulations, pedestrian detection, and area reconstruction while talking about related works in literature. Chapter 3 gives an overview of the framework. Chapter 4 is on video stabilization, pedestrian detection and tracking where three methods for detection are discussed. In Chapter 5, three methods for image segmentation and navigable area determination are compared. The resulting navigable segments are then passed to the navigable area reconstruction subsystem, as described in Chapter 6 including a method for construction of 2D meshes from 2D monochrome images. The included crowd simulation is described and the user interface is demonstrated in Chapter 7. Chapter 8 concludes the thesis by showing and discussing the results for crowd simulation and discussed possible feature works related to limitations of the framework.

Chapter 2

Background and Related Work

2.1 Pedestrian Detection and Tracking

In computer vision, pedestrian detection is performed in its most basic case using Histogram of Oriented Gradients (HOG), which is also called Dalal-Triggs detector [8]. Models for pedestrians, which contains feature descriptors that form the postures of pedestrians, are constructed by applying feature extraction over the image. Then the aforementioned models are classified as pedestrians in the image using a sliding window approach. This method is able to accurately detect and locate pedestrians in the given image. We utilized this method as part of our pedestrian detection work, therefore in Section 4, we briefly explain and demonstrate it on our video data.

Breitenstein et al. [9] use HOG in order to create a framework for multi-person detection using a monocular, uncalibrated camera. They utilized HOG as a detector in which they used particle filtering for tracking. Each tracker is trained individually in real-time for the features of detection it is assigned, which is said to provide robust tracking performance in cases where occlusions occur or multiple targets interact. Our framework and theirs both use HOG for pedestrian detection, but we also used neural networks that are capable of

distinguishing pedestrians with high accuracy. The other main difference is in tracking where they used a particle filter and we used Kalman filtering.

In recent years, developments in the area of neural networks have flourished. Most noticeable work is being done in the area of image processing, specifically image classification [10]. In order to classify the given image, as cat or dog for instance, researchers construct a neural network that learns the hidden features in the image after training with a high number of example images. In our case, we not only need to classify if there is a pedestrian in the image but also locate it accurately. For this purpose, Recurrent Convolutional Neural Networks (R-CNNs) have emerged and proved to have high accuracy in both detection and localization of not only pedestrians but also arbitrary, yet distinguishable objects.

R-CNNs are advanced versions of Convolutional Neural Networks (CNNs). R-CNNs takes the given image as input and outputs bounding boxes that define the locations and regions of the detected objects. The first part of the network is to investigate the image for potential regions that contain object(s). The simplest way to do this is to apply a sliding window approach similar to HOG, but this method is not feasible in terms of performance. Girshick et al. [11] use Selective Search [12] that has a hierarchical approach. A version with improved performance and accuracy is proposed in [13], named “Fast R-CNN”. “Faster R-CNN” proposed in [14] replaces region proposal part with a separate network that gives an even better performance.

Other than R-CNNs, there are one-shot approaches who runs through the network only once. YOLO [15], for instance, gives the whole image to the network (after resizing) and predicts bounding boxes for each tile in the uniform image grid. Another one-shot approach is Single Shot MultiBox Detector (SSD) [3], which uses a similar approach but yields better accuracy. We utilized SSD in our detection framework and it is going to be further explained in Section 4.

2.2 Data-Driven Crowd Simulations

Our framework can be regarded as a data-driven simulation as it uses real-time detection and tracking data of pedestrians in the given video feed. When we look at the data-driven simulations, however, the extracted information of pedestrians is not used for projecting them on the simulation environment but investigating the underlying properties of the crowd they form.

For instance, in the sense of collision avoidance, Guy et al. [16] use personality traits to determine a mapping of behavior parameters to related collision avoidance parameters. Similarly, Turkay et al [17] utilize information theory for behavior models of the agents. Bera et al. [18] introduce personal spaces that are related to pedestrians’ observed personalities. Personalities are obtained from Reciprocal Velocity Obstacle (RVO) parameters and the artificial agent, a robot, is able to make predictions about trajectories of pedestrians.

Some methods use various techniques to synthesize artificial crowds simulations, where some of them are interactive. Musse et al. [19] extract the trajectories of pedestrians and determines the velocities of artificial agents in the simulation environment. Additionally, agents are able to react to panic-inducing situations by heading to the nearest exit. Another data-driven approach for generating a crowd is introduced by Kim et al. [20], which learns from trajectories in a fully automatic manner to generate an adaptive, dynamic crowd usable in various environments. In contrast to our work, none of the approaches projects the agents directly onto the simulation but they rather synthesize a crowd out of them.

Lerner et al.[21] build a trajectory database from trajectories of real agents, which are obtained manually from the input video. In the reconstructed scenario, the virtual agents form a query to the database according to their current surroundings. The response from the database found using heuristic functions, directs the virtual agent away from possible collisions with other agents. Başak et al. [22] learn emotion contagion parameters from the individual agents in a given input video which aims to improve the behavior of virtual agents in the

recreated scenarios in the sense of realism.

Jablonski et al. [23] use pedestrian flow data to evaluate the similarity between synthetic recreation of the crowd scenes. Similarly, we used Kalman Filter when we are tracking the pedestrians. However, they create the environment manually using a 3D design tool whereas we generate it automatically. Another approach to analysis is given by Charalambous et al. [24] where they present a framework that uses machine learning techniques, including Principal Component Analysis (PCA), to detect errors in tested pedestrian behavior data relative to reference data.

2.3 Area Reconstruction

The first part of our area reconstruction approach is based on image segmentation where the given image is partitioned into meaningful regions. For this purpose, Achanta et al. [25] segment the given color image into distinct superpixels which search for candidate pixels in a limited region in contrast to k-means approach. Size of the search region is determined by the given superpixel size. Each pixel in the image is represented by both their color (in LAB space) and spatial features that create a 5-dimensional feature vector where it is possible to give different weights to features.

There are also density-based approaches that search for the best cluster definitions according to their connectivity with their neighbors. Ester et al. [26] search for cluster cores that accumulate other elements where Campello et al. [27] search for the most stable clusters in a hierarchical structure. Meanshift algorithm [28] finds local maxima in the feature space using kernel density estimation and calculates a mean-shift vector, proportional to the density gradient that leads elements towards it. We tested and compared the approaches proposed in [26], [27] and [28] (see Chapter 5). There are also depth estimation methods to obtain the ground regions such as the one proposed in [29]. In their work, ground pixels are identified using semantic labeling. Then they use gradient boosting technique to estimate

the baseline depth for pixels and use RANSAC for plane parameter approximation which gives the ground regions together with estimated depth. Their method is prone to errors when depth is above 79 meters, which decreases its scalability. As we expect to work with videos with arbitrary depths, this method cannot be applied directly.

Identification of navigable regions leads to deciding their location in 3D Euclidean space, which requires some understanding of the scene of interest. One of the ways is to approximate the horizon in the given image; which gives us a good understanding of our camera configuration and placement relative to the ground plane. For this purpose, Li et al. [30] find the horizon and a third vanishing point using intersections of Hough lines in the image. Another approach by Zhai et al. [31] use deep convolutional networks to find a suitable horizon that is compliant with the third vanishing point, found using the lines segments similar to the previous method. Trocoli and Oliveira [32] generate a histogram of line segments by their angles and uses the peaks when searching for vanishing points. These methods rely on the existence of horizontal lines when finding the vanishing points, which mostly exists in man-made structures. In our case, we don't guarantee to have a visible structure, such as a building, in our scene that is enough to determine the vanishing points.

In contrast to previous methods, some studies such as [33], [34] and [35] used the pedestrian features when extracting the vanishing points. They treat the obtained pedestrians as vertical poles where their postures intersect at third vanishing point and their trajectories at other vanishing points on the horizon. Liu et al. [34] use the pedestrian postures to find the nadir point in the image which is then used to find the optimal definition for focal length by enumeration, to maximize the similarity with real-life pedestrian height distribution. Similarly, Brouwers et al. [35] take into account the height distribution when determining the tilt angle of the camera. Another method by Jung et al. [36] calibrates the camera while estimating the horizon using postures to find the normalized height of the tracked pedestrian. We also use pedestrians in the image to calibrate the camera, but we also utilize the line features that make our framework more applicable to arbitrary scenarios where pedestrian data is limited and noisy.

Given the vanishing point information, the navigable regions can be viewed from a frontal view for accurate modelings, such as a blueprint. This process is called *metric rectification*. Liebowitz et al. [37] apply metric rectification to perspectively distorted images and recovered the line and angle properties observed in real-world. This method used the concept of circular points, calculated from image lines, which is explained in more detail in our work at 6.2; when constructing the matrices that will correct our images. Liebowitz et al. [38] use metric rectification to fully 3D reconstruct the visible regions in the given image. Chaudhury et al. [39] use line properties to find two vanishing points which are then used to affine rectify the user photos on the Internet. However, affine rectification corrects the perspective distortions only partially: they do not recover the angles in the real world. Bose and Grimson [40] track moving objects in the scene, such as pedestrians, to find the horizon and following the approach of circular points [37], metric rectified the ground plane. In our framework, we combine the works of [40] and [39] to find three vanishing points in the scene, then metric rectifies the navigable regions in the image using circular points found from pedestrian trajectories and horizon orientation.

There are also methods for full 3D reconstruction of the scene including the vertical structures such as walls. Manual approaches such as Bulbul and Dahyot [41] use social media location sharing data to populate cities that are 3D built using OpenStreetMap [42]. Each check-in location is used for placement of each agent which is supported by matching the features in the photos with ones in the references shots, using scale-invariant feature transform (SIFT) [43]. Iizuka et al. [44] manually annotate the boundaries of 5 polygonal regions: left, right and rear wall, ground, and ceiling in a single image, then determines the foreground objects using a lasso tool aided segmentation to reconstruct the scene. Their framework enables the user to have a walkthrough in the scene and observe semi-automatically placed foreground objects. Zhang et al. [45] automatically reconstruct the scene based on epipolar geometry of multiple views. Single view approaches, such as [46] by Hoiem et al. and [47] by Saxena et al., reconstruct the scene fully automatically.

2.4 Augmented Crowd Simulations

With the increase in processing power of smartphones with single and even multiple cameras, it become possible to enhance the videos in real time. Therefore, augmentation of dynamic objects and agents into real videos draw attention because rendering realistic interactions with the environment is a challenging task. For the simplest case, Thalmann et al. [48] create the simulation environment manually and inserts a virtual character with basic behavior. Fernandez et al. [49] use Natural Language Processing (NLP) to generate and control virtual agents that interact with their surrounding environment. The behavior of agents is determined by a fixed set of rules in Situation Graph Tree structure, combined with the interpretation of actions of real agents via fuzzy logic.

Narahara and Kobayashi [50] use real projectors to project pedestrians over real architectural models to obtain a crowd walks through on them. Using real-life markers, Zheng and Li [51] create and augment interactive virtual crowds that are able to perform various group behaviors. Oliver et al. [52] use virtual reality (VR) in a collision avoidance scenario between a participant and a single virtual agent to investigate the effect of VR on visibility to avoid collisions. Additionally, several locomotion interfaces are introduced to participants to assess their usability.

Baiget et al. [53] augment multiple agents, which are reactive to their surroundings, into the simulation environment in real-time. They generate the environment using a calibrated static camera. Our approach is able to do this automatically on an arbitrary video feed, which means calibration information is not necessary. Additionally, we used a better collision avoidance technique rather than basic distance evaluation. Their work includes non-human dynamic obstacles such as cars, where our detection approaches based on neural networks can be used as a baseline to add more diverse objects. Rivas et al. [54] describe a framework for combining artificial agents with real pedestrians, which is similar to ours. They used background subtraction and SVM classification to detect pedestrians. Real pedestrians are then inserted into a simulation environment where additional agents are added on top.

Chapter 3

Framework Overview

Our framework provides an interactive crowd simulation in Unity game engine [55] which is augmented on a surveillance video. Real pedestrians are simulated with artificial agents over the navigable regions in the video. To reconstruct the navigable scene within the input video, we pre-process it using computer vision techniques that are included in but not limited to OpenCV library [56]. The simulation is real-time using C# in Unity where the preprocessing is done off-line in Python 3. Our framework has multiple subsystems (see Figure 3.1). The main stages of our framework are as follows:

- We take a surveillance video or a video taken with a static camera as input. The video features might be enough for reconstruction of the scene within, but videos that include multiple active pedestrians are recommended. The input video is stabilized on the fly by calculating the optical flow and adjusting adjacent frames with homography accordingly.
- In order to detect the pedestrians in the stabilized frame, conventional Histogram of Oriented Gradients (HOG) [8] or neural network-based detectors such as *Inception* [4] and *MobileNet* [57] are used. The user is provided with the implementation of each method together with their relative parameters.
- The detected pedestrians are tracked using a Kalman filter. The tracked

pedestrian locations are recorded on a text file in a MOT [58] compatible format for each frame. The framework also reports each pedestrian’s posture based on the foreground pixels of the current frame.

- The first frame of the video is used to find navigable regions in the video. The frame is segmented using one of the provided algorithms: DBSCAN [26], HDBSCAN [27] or Meanshift [28]. The navigation trajectories of real pedestrians in the video are used to identify the segments which will also be navigable for artificial agents. Navigable segments are reported as white in the output black-white image.
- In order to reconstruct the navigable areas resulted from the previous step, the vanishing point information from the first frame is extracted. The configuration of such points is used to identify the structure of the scene, which determines the 2D blueprint of the navigable areas. This “blueprint” is approximated by applying perspective correction on the segmented image. The camera to be placed in Unity is located using the correspondence of the blueprint model on Unity’s XZ plane and OpenCV’s XY image frame corners. This relation is realized by an iterative perspective-n point solution. In this process, the imperfect blueprint is corrected to match its real-life counterpart and converted to a 2D mesh. The camera configuration is reported in form of text file, ready to be read by Unity.
- 2D model from the previous subsystem is placed together with the camera in Unity. Camera loosely represents the original one in the video, by matching the navigable regions from the video with the projection of the model on its image plane. The crowd simulation model in Unity is based on RVO and navigation meshes with clearance for collision avoidance. Each pedestrian detected in Unity is projected onto the placed navigable area and represented using a dummy model. The user can insert realistic looking agents in the scene, which interacts with projected agents through collision avoidance. The height of artificial agents is adjusted according to the perceived height of projected agents in the simulation environment.

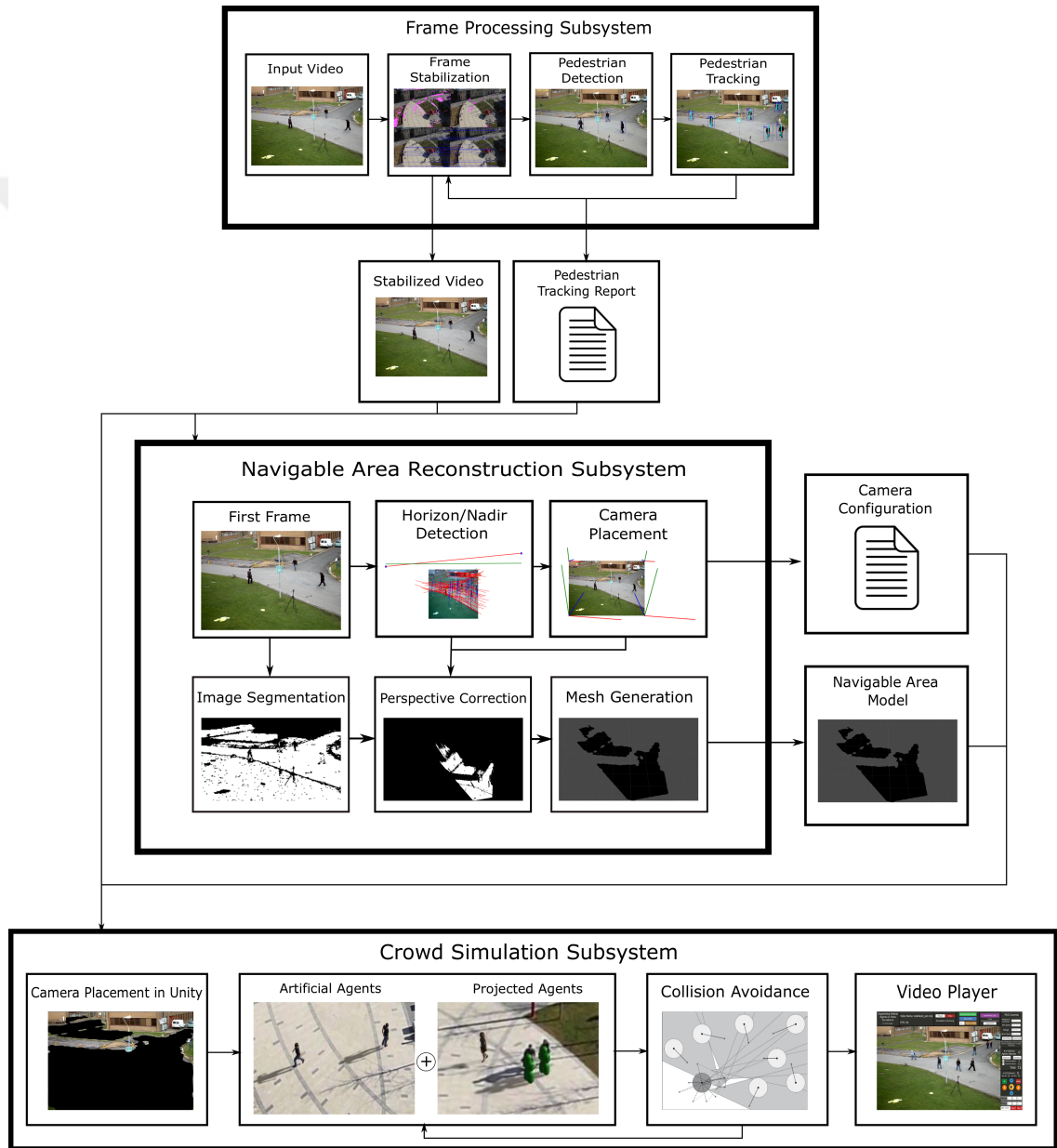


Figure 3.1: The proposed framework.

Chapter 4

Pedestrian Detection and Tracking

The first step of our framework is detection and tracking of the pedestrians in given video feed. For this task, we assume that the given video is static, which means the position or orientation of the camera is not altered. However, there can still be minor disruptions during recording due to human error, wind and so on. Therefore, in order to make sure our video is perfectly static as it is assumed; it needs to be stabilized first. The stabilization process is described in Section 4.1.

As we aim to augment the navigation of real people in the video, idle pedestrians can be ignored; as they don't contribute to the overall simulation much. For this reason, and to decrease the time spent per frame, we only search for pedestrians in the areas where movement occurs. In order to detect such areas, we use background subtraction [59]. Background subtraction is suitable for our purpose as we assume, and make sure by stabilization, that our camera is static.

We use different approaches for pedestrian detection, as discussed in Sections 4.2 and 4.3. These include sliding-window of HOG and one-shot (single run of neural network) of SSD; where two networks, *Inception v2*, and *MobileNet*, are available. We adjust their related parameters individually for each video.

Every method generates bounding boxes for detected pedestrians but as detections can be noisy, they cannot be completely trusted. Also, in order to reflect the real life velocity of the detected pedestrians to the simulation environment, their consecutive positions in the video needs to be known. For this reason, we track the pedestrians using Kalman Filter, which uses both spatial information and histogram of the proposed detection for assignment. In the output of this framework, only consistently tracked pedestrians are included in order to reduce false positives. In addition to their positions, postures of the pedestrians are also extracted using foreground areas. Their effect in simulation is minimal, but they are to be used in navigable area reconstruction (see Section 6).

4.1 Stabilization of the Video

In order to stabilize the whole video, each frame should be stabilized individually. For this purpose, we calculate the optical flow to the current frame from a reference frame, using Lukas-Kanade algorithm [60]. The optical flow will be used to “warp” our current frame so it almost matches with the reference frame. The first frame is used as a reference through the stabilization process.

In the first step, the frame to be processed is converted to gray-scale, as brightness values are used for optical flow calculation. Then, an image pyramid is constructed; it contains arbitrary scaled versions of the input image. This step is necessary, as Lukas-Kanade algorithm fails in cases where the motion is large. By decreasing the size of the input image, motion size is also decreased. In the second step, features from the image need to be extracted for optical flow calculation. We use the approach described in [61] for this purpose with parameters determined experimentally. The detected points are used for optical flow calculation as follows. Given the pixel value for a feature at (x, y) in reference image as $I(x, y)$, the location of the feature in the current image is

$$I(x, y) = I(x + u, y + v), \quad (4.1)$$

where u and v are the displacement in x and y directions, respectively. To determine u and v :

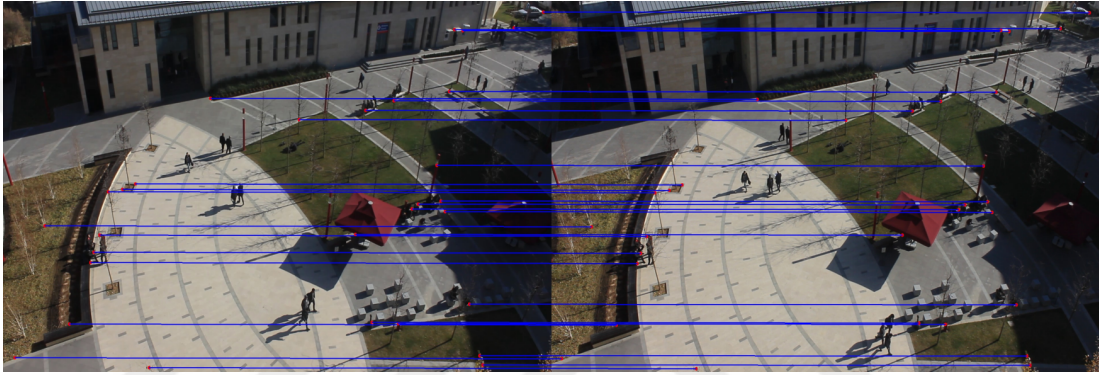
$$0 = I_t(x, y) + \nabla I \cdot [u \ v], \quad (4.2)$$

where ∇I represents the spatial gradient. The optical flow direction is the vector with unknown u and v values. Using Lukas-Kanade algorithm, we solve this equation to extract the optical flow vector. However, the point correspondences cannot be determined in some cases and such points are discarded. We apply forward-backward validation to find the re-projection error for features in the image.

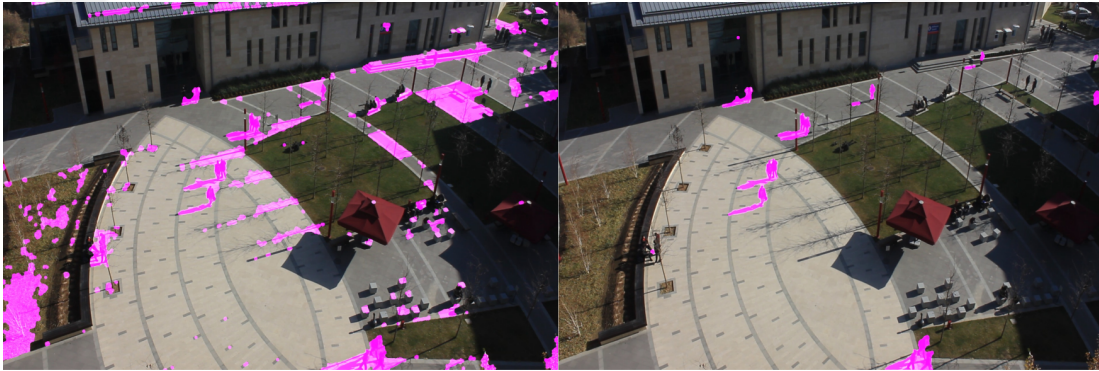
Using the remaining point correspondence between frames, as seen in Figure 4.1 (a), the homography matrix is calculated using normalized Direct Linear Transform [62]. Given x_i as points in the reference frame and x'_i for the target frame, the homography maps them as $x'_i = Hx_i$. The homography matrix is multiplied with the current frame to closely match the reference frame. After stabilization, background subtraction using [59] is applied to the image. The importance of stabilization for background subtraction can be observed in Figure 4.1 (b). Each foreground blob is then passed to the detector for searching pedestrians.

4.2 Pedestrian Detection using HOG

HOG [8] for detection is based on distinguishing objects from others according to their appearance, which is their pose in the image. Their pose is specified by their feature descriptors, which are calculated from gradients in the image. Given an input image containing a pedestrian, the descriptors are calculated after dividing



(a)



(b)

Figure 4.1: The importance of stabilization for background subtraction. (a) the “good features” from the reference and target frames, together with their correspondences. (b) For background subtraction, it is important to stabilize the frame first; background pixels can be misclassified as foreground, which are shown as pink (left). After stabilization, only pedestrians and shadows are classified as foreground (right).

the image into uniform cells. We calculate the descriptors by sliding a filter mask across all cells in the image, where a cell can be contained by multiple filters, which results in a feature vector, called *histogram of gradients*. Each cell holds a histogram including orientation and magnitude of pixels in its region.

After calculating the feature descriptors for each cell, a detection window is scanned, by again a sliding-window approach. The detection windows have arbitrary scales, therefore they create a pyramid structure. Each region is fed to a linear support vector machine (SVM) that classifies if it contains a pedestrian. The SVM has been trained before using positive and negative sample images on pedestrians, where pedestrians are mostly seen in standing position. In case where multiple pedestrians are detected in a region, non-maximum suppression (NMS) is applied to get rid of the redundant detections. This method has been widely used for pedestrian detection. We use the HOG+SVM+NMS approach in our videos by adjusting parameters related to each.

4.3 Pedestrian Detection using Neural Networks

Aside from classic HOG for pedestrian detection, we use pre-trained neural networks for detection. One-shot detection networks are able to detect pedestrians with high accuracy in acceptable time. We use Single Shot Multibox Detector (SSD) [3] for this purpose. A general structure of SSD is provided in Figure 4.2. The image is given as input to the network. The first part of the network is called the base network and it includes convolution and pooling layers. The fully connected layers at the end of the original base network are also replaced by convolutional layers.

After the base network is constructed, each feature map is filled with so-called *default boxes* that are bounding boxes with predefined ratios and sizes. One of the bounding boxes is limited to the cell in feature map, where others have arbitrary

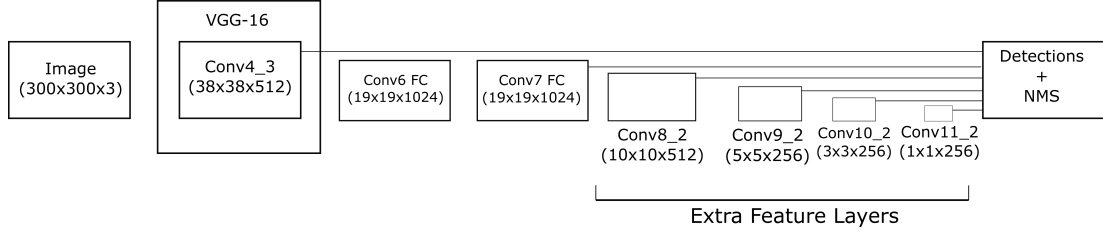


Figure 4.2: The simple visualization of SSD [3] layer structure using VGG-16 [2] as base network.

ratios. Each default box calculates a score for every class to determine the types of objects in that region. By using multiple default boxes with varying shapes, SSD is able to combine each prediction to detect objects with arbitrary orientation and sizes. SSD includes additional default boxes when processing some feature maps. After base network, SSD includes convolutional layers with gradually decreasing resolution. This results in smaller feature maps where large objects start falling into less number of tiles. Therefore, SSD is able to detect larger objects without compromising smaller ones, as they are detected in different resolutions.

One possible advantage of using a neural network over HOG is: neural networks can be used to detect arbitrary objects such as cars, dogs that can be incorporated into the simulation without requiring additional detectors. This makes the framework scalable for more complex scenarios. There are two base networks available for SSD, *Inception v2* and *MobileNet* which are going to be explained briefly in upcoming sections. The trained versions of the networks are provided as part of OpenCV Tensorflow API [63], [64], which are open source.

4.3.1 *Inception* Network

Inception network model [4] is based on its unique structure consisting of building blocks called *inception modules* (see Figure 4.3). Each module includes parallel networks for convolution and max-pooling, which are combined and fed to the next module. However, the computational cost is high as convolutional layers add more and more depth to the output. In order to prevent this issue, each layer in Inception module, except 1×1 convolutions, is connected to a 1×1 layer

which lowers the size of the outputs by reducing their depth.

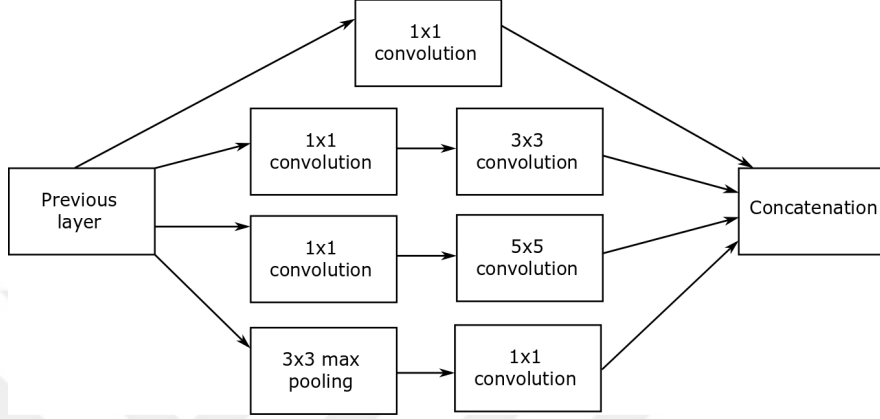


Figure 4.3: The inception module, as described in [4].

In our framework, we use an improved version of the network, called *Inception v2* [65]. By replacing 5×5 convolutional kernels with 3×3 kernels to reduce the number of calculations and combining it with batch normalization [66], in order to reduce the variance shifting in hidden layers, it aims better performance than the previous iteration.

4.3.2 *MobileNet* Network(s)

MobileNet(s) aim acceptable performance with low cost on mobile devices where available hardware is an issue. The principal of the network is using separable convolutional structure including a depth-wise convolutional layer where a single filter used for each channel followed by a generic 1×1 convolutional layer [57]. The depth-wise layer is used to decrease the computational cost but as it does not generate any new features from the combination of multiple channels, it needs to be used together with a convolutional filter. Using this structure, the overall computational cost is reduced.

4.4 Pedestrian Tracking

In our framework, we cannot solely rely on the detection boxes of the pedestrians as detectors are not completely reliable and do not provide any information about if two detected pedestrians are the same person. Even though we are not concerned with representing real pedestrians in our resulting video visually and personally accurate, tracking them is necessary: as their interactions with artificial agents are based on their velocities in collision avoidance. Therefore, their trajectories need to be determined as accurately as possible. An example tracking can be seen in Figure 4.4 For this purpose, we utilize Kalman Filters [67].



Figure 4.4: Using Kalman filters between consecutive frames, we are able to predict the next location of the tracked pedestrian and correct it according to detection in the next frame.

At every frame, each detection box of the pedestrians is associated with a tracker object with a model that consists of spatial position and visual features of the detection window, which are fed to a Kalman filter. We perform the association in a greedy manner where each detector is assigned to the tracker with the lowest cost. For each detector not assigned to a tracker, a new tracker is generated. Any tracker that is not associated is considered to be removed. We calculate the cost used in the association process according to

$$cost(tr, d) = \alpha \times cost_{vis}(tr, d) + (1 - \alpha) \times cost_{pos}(tr, d), \quad (4.3)$$

where α is determined experimentally. The positional cost represents the distance between prediction from the Kalman Filter to the center of the detected bounding box. The distance is normalized according to the predefined threshold, that gives us the positional cost. If the calculated distance is higher than the threshold, the cost is assigned as one.

The visual cost is calculated as the distance between the visual features that are associated with the tracker model and detection window features; similar to the positional cost. The histograms of the detection area in LAB color space are taken as features. Upon association with a detection, the tracker’s visual model is updated by blending, which is also reliant on an experimentally determined parameter. The importance of using visual cost aside positional one is seen when two distinct pedestrians come very close to each other, the association process might fail due to positional similarity. More importantly, when a tracker is not associated with a detection for few frames, due to occlusion, for example, the visual cost might help to locate it again.

In cases where a tracker is left unassigned, its state is checked for removal. The state diagram in Figure 4.5 is used to decide what needs to be done with the tracker. A new tracker is created when a detection is not associated with any tracker. However, false detections can occur because of noise. In order to decrease the false positives, the tracker is not reported as a pedestrian until it is associated to multiple detections over the course of consecutive frames. Otherwise, it is considered as a false detection and removed.

If an active tracker stays unassigned for a certain number of frames, it becomes deactivated. A deactivate tracker is considered as a valid pedestrian where its position is predicted by the associated Kalman filter, but it is not reported. Tracker becomes active again when a suitable detection is found, otherwise, it is removed.

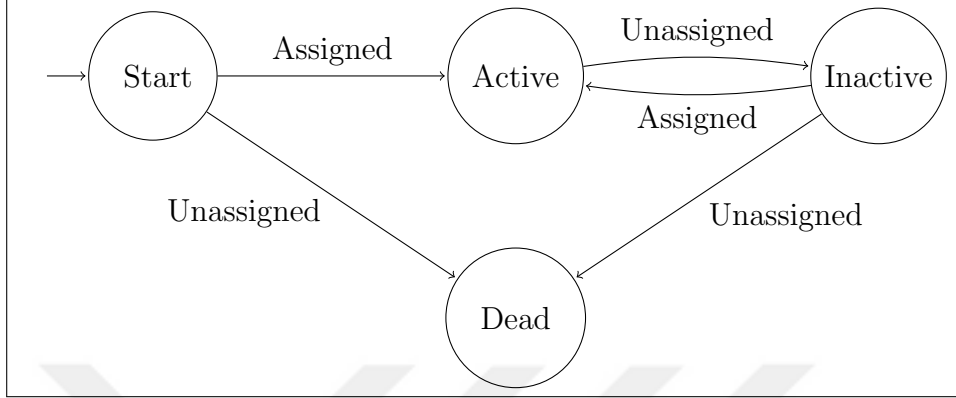


Figure 4.5: State diagram of the tracker. Only active trackers are reported as pedestrians in the output.

In our framework, tracking is performed at every frame rather than once in a few frames. As the position of real agents is crucial for realistically simulating them with collision avoidance in our scene, constantly updating their trajectory is required. Heavily relying on predictions of Kalman filter would result in invalid positioning and late correction of the filter predictions might cause sudden, unrealistic changes.

4.5 Posture Detection

The tracked pedestrians are represented as bounding boxes where the pedestrian can be seen in arbitrary postures. Taking the bottom center point of the bounding box is good enough to determine the feet position in the image, which is then projected on the scene as explained in Chapter 7. Yet in order to aid the reconstruction of the navigable area, we extract the postures of the pedestrians as major axes between their head and feet in spatial space. Postures including head and feet can be used as cues for camera calibration, which is investigated in [33].

Our approach is similar to the one described in [33]. First, a valid detection of a pedestrian, associated to an active tracker, masked with the foreground mask generated by the background subtraction algorithm (see Section 4.1). The

detection box is converted to the LAB color space and L (luminance) channels are thresholded according to mean and variance. From the remaining blobs, the largest one is selected after dilation and erosion. The centroid and orientation of the largest blob is found using image moments [68], which are used to define the principal axis. The head and feet positions are calculated as endpoints of the principal axis. Unlike Zhao and Nevatia [33], we report all phases of the legs of pedestrians, where they can be open while navigating.

In order to combat bad postures resulting from shadows and multi-pedestrian detection boxes, some constraints are introduced for the posture extraction process. The angle of the orientation, the ratio of principal axis length with the diagonal length of the bounding box and its distance to the center of the box are used as constraints. Example postures can be seen in Figure 4.6, each row includes examples from a different video.

4.5.1 Results

We evaluated the performance of various detectors and tracking combinations by testing them on our videos described in Chapter 3. All videos contain a static camera but each has different resolution, view angle, and distance to the navigated area in real life. Therefore, parameters related to detectors and tracker are adjusted individually for each video in order to improve performance. The user can also give parameters to the framework to find the best match for his/her own use-case. Our evaluation is performed in two parts for each video: Quantitative evaluation includes comparisons with ground truth detections in terms of accuracy and qualitative results include visualization of resulting bounding boxes around tracked pedestrians.

4.5.1.1 Quantitative and Qualitative Tracking Results

We implemented each detector and tracker in Python language using OpenCV library [64], [63]. We obtained ground truth labels for PETS09-S2L1 from PETS



Figure 4.6: Example postures from different videos. Top three rows contain detections and corresponding foreground masks. Blue and green dots represent the rough position of the head and feet respectively. When legs are open: green points is the bottom point of the principal axis. Last column contains invalid postures, shown with red dots. In the middle row, rightmost posture is mistaken as a valid one, even though it contains multiple postures. At the bottom, each pedestrian together with their detection boxes (blue), unique tracking ID, posture line (cyan) and approximated direction arrow (red) is shown.

dataset [69]. Because we recorded the other two videos, we construct the ground truth manually. Our videos are expected to be more challenging than PETS video we obtained from the database, as the camera in Video 1 and 2 monitors the navigable area from a very far position compared to PETS and pedestrians, which are dense, are not always seen in vertical posture relative to the view angle. Adding static and dynamic shadows to the mix further increases their difficulty.

For quantitative evaluation of our detection and tracking, we used metrics from [70] which contains recall, precision, multi-object tracking precision, and accuracy. Recall is the percentage of correctly identified pedestrians (true positives, TP) to the total number of pedestrians in the video (TP + false negatives, FN), where precision is the ratio of relevant detections (TP) to all detections (TP + false positives, FP). As seen in Table 4.1, HOG based detector gets higher precision than RNN based solutions yet its recall is lower.

RNN solutions having a higher recall is not surprising, as more objects in the scene are detected than HOG. As RNN based solutions are designed to recognize multiple classes of objects from COCO dataset [71] including cars, animals, and cell phones. Detected objects that are not classified as pedestrians are ignored automatically, yet some objects including street lamps, traffic cones or everything that has a vertical posture can be mistaken as a person. This is related to the confidence threshold for the detector; decreasing would increase the number of false positives as arbitrary objects can be considered as pedestrians. Increasing it would force the detector to only report when its highly confident that the object detected is a person, but this might cause it to miss some pedestrians.

For our purposes, getting a higher recall is more crucial than precision as high recall indicates we were able to detect most of the pedestrians in the video, which will enable our artificial agents to avoid them. A high precision indicates we don't identify arbitrary objects as people much. A low precision might cause the navigable area to be crowded by "ghosts" that are simply noisy detections which decrease the available area for our artificial agents to traverse. A low recall, however, will cause our agents to go pass pedestrians in the video; which is against the purpose of our simulation. In the end, as the detectors and tracker

are heavily reliant on the parameters, it is up to the user to adjust them to suit their needs.

Other two metrics we used are multi-object precision (MOTP) and multi-object accuracy (MOTA). MOTP is used to measure the mean dissimilarity between correct detections and ground truth. As it represents dissimilarity, getting a low MOTP would indicate good performance. As it is seen in Table 4.1, RNN solutions have lower MOTP than HOG. On the other hand, MOTA [72], [73], brings FNs, FPs, and mismatch errors (MMEs) together in:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + MME_t)}{\sum_t GT_t}, \quad (4.4)$$

where t is the frame index, GT is the number of ground truth objects and MME is the mismatch error. Mismatch error occurs when a single object is given multiple identification numbers. In our tests, *Inception* seems to have the lowest MOTA which can be explained using its recall and precision. Having a high recall but low precision would indicate that there is a very high number of detections where only a certain portion is relevant, which is expected with a high number of classification errors; decreasing MOTA. In some results, MOTA has a negative value. This is because detection errors are higher than the number of pedestrians [70]. Overall, *Inception* performs better than *MobileNet*, except for timing. *MobileNet* aims acceptable performance on limited hardware, which explains why its frame computation time is much lower than *Inception*. Still, *MobileNet* is much slower than the HOG base detection.

Figure 4.7 shows example outputs of the tracking subsystem from PETS video. The stabilized image at the top, where HOG, *Inception* and *MobileNet* results are ordered downwards respectively. The most noticeable difference between detection systems, particularly in the frames shown, is that RNN based detectors locate more pedestrians. This is supported by their high recall. However, with better adjustment of parameters, RNN solutions can surpass traditional HOG in

terms of precision as they already do with recall. Besides, *Inception* classifies non-pedestrian objects as pedestrians whereas *MobileNet* does not (see Figure 4.7).

Table 4.1: The quantitative results of our pedestrian tracker. The experiments were performed on a personal computer with Intel®Core™i7-4500U CPU @1.8 GHz, 8 GB RAM and NVIDIA 740M.

PETS09-S2L1 (768×576 @15.0 fps)

Method	Recall	Precision	MOTA	MOTP	Frame computation time (sec)
HOG	81.8	76.8	55.5	0.314	2.0
<i>Inception v2</i>	92.8	47.6	-10.5	0.241	8.7
<i>MobileNet</i>	91.4	63.2	36.8	0.244	5.3

Video 1 (1280×720 @30 fps)

Method	Recall	Precision	MOTA	MOTP	Frame computation time (sec)
HOG	18.1	41.4	-7.8	0.400	4.1
<i>Inception v2</i>	42.9	31.8	-50.2	0.343	13.2
<i>MobileNet</i>	30	28	-48	0.334	8.7

Video 2 (1920×1080 @23.976 fps)

Method	Recall	Precision	MOTA	MOTP	Frame computation time (sec)
HOG	44.8	65.8	21.0	0.333	5.2
<i>Inception v2</i>	74.6	45	-17.6	0.324	16.0
<i>MobileNet</i>	58.7	44.1	-17.1	0.325	10.2

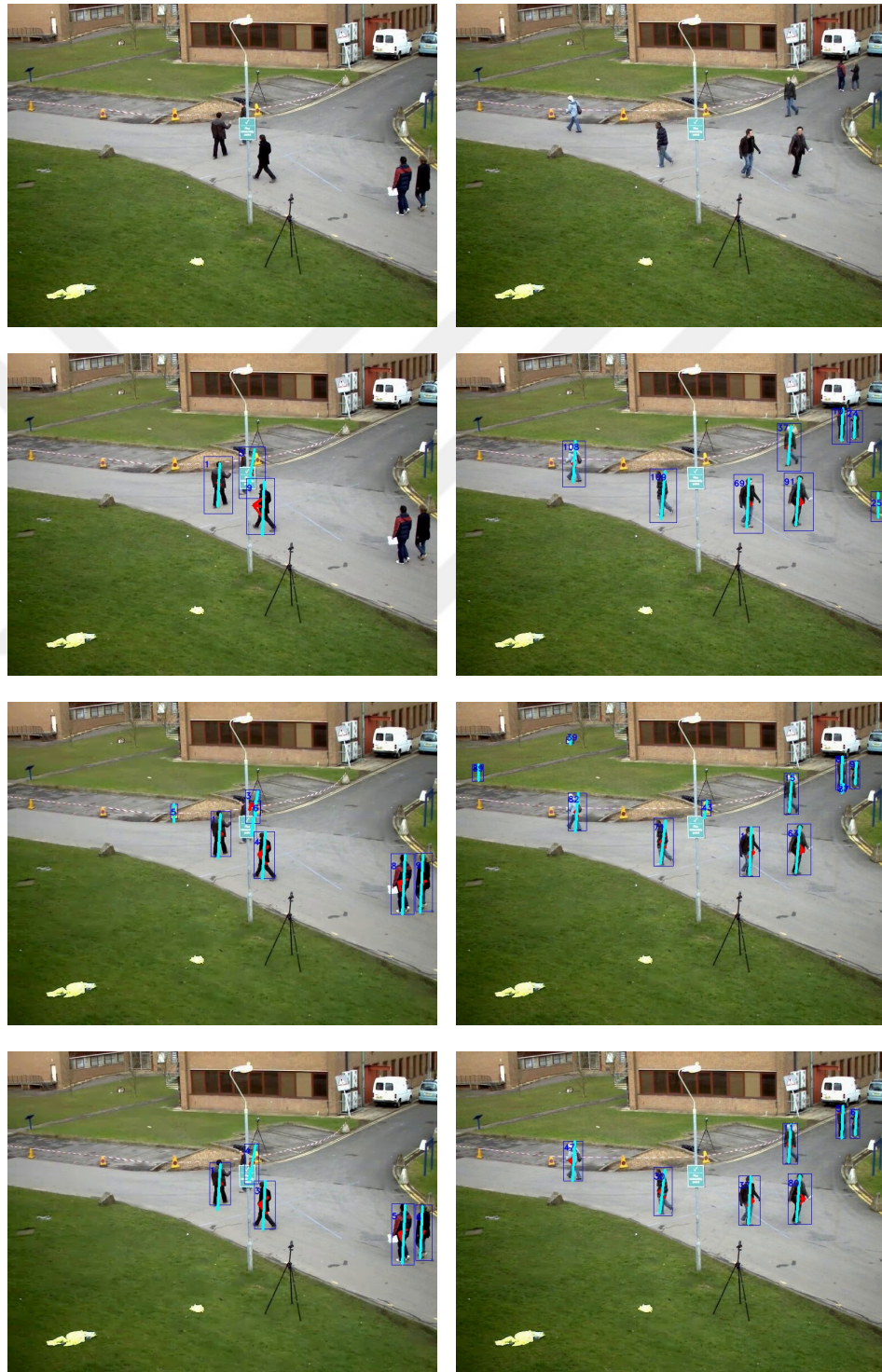


Figure 4.7: Example pedestrians postures from PETS video. The stabilized image at top, HOG, *Inception* and *MobileNet* ordered downwards. More can be found in the appendix and electronic copies of tracking videos.

Chapter 5

Image Segmentation for Navigable Area Extraction

After detecting and tracking the pedestrians in the given video, they are going to be simulated together with artificial agents in the scene. For this purpose, the scene must be reconstructed as a 3D environment so both types of agents can navigate and interact in it. In order to construct such navigable areas, they need to be extracted and identified from the input video. In our framework, these are done using image segmentation techniques. Image segmentation is the process of separating the given image into multiple, meaningful clusters based on spatial and channel (color) information of each pixel. Even though we are working with videos, as our cameras are assumed to be static, it is enough to extract regions from a single image from the video. This image is chosen to be the first frame. Considering the videos of our interest, the segmented regions might contain various, distinguishable textures that can be directly associated with navigable areas in real life; such as grass, stone pathways, and roads. However, we wish to extract arbitrary textured navigable areas rather than fixed assumptions. This makes our approach generic and scalable to various scenarios.

After extracting regions from the frame, they need to be classified one-by-one as if they are navigable or not. The "navigability" of regions depends on the

pedestrians in input video. A particular region is navigable, if the percentage of total frames that region is navigated by at least one pedestrian, is above a threshold. However, such labeling is done per-region rather than simultaneously for all similar regions. Therefore, for example, labeling a region of grass as navigable has no effect on other regions that are also grass. To identify which regions the pedestrians' have traversed, we used their posture information from Chapter 4.

For segmentation, we utilized different three techniques: Density-Based Algorithm for Discovering Clusters (DBSCAN) [26], Hierarchical DBSCAN (HDBSCAN) [27], and Meanshift [28]. In the following sections, each method will be described briefly and then compared. We define the features for the image of interest as combination of spatial and color features of each pixel. We test every method with multiple color channels.

5.1 Meanshift

Meanshift algorithm [28] clusters a given data by leading data points, in our case pixels, towards the local maximum via kernel density estimation 5.1. Using the kernel function (K) on each data point x , where h is the bandwidth and n is the number of points, an overall density distribution is calculated. Then, a mean shift vector is calculated for each point using 5.2 where $N(x)$ are the neighbors of x .

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right), \quad (5.1)$$

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x - x_i) x_i}{\sum_{x_i \in N(x)} K(x - x_i)} - x. \quad (5.2)$$

Mean shift vector represents the direction towards the local maximum in the sense of density, as it is proportional to the density gradient. After multiple iterations, the algorithm stops when the magnitude of the mean shift vector

becomes fractional; thus points reach convergence and form a cluster. Meanshift relies on a single parameter for bandwidth, h in the formula, which requires to be determined empirically. We used the bandwidth estimator of sklearn, a software machine learning library [74], which determines the bandwidth from samples of input data.

5.2 DBSCAN

DBSCAN [26] works by clustering the given data according to their connectivity. Such connectivity is determined by a number of definitions, which depends on two parameters, the number of neighbors (m), and the distance parameter (ϵ).

- *Core sample*: a sample that contains at least m neighbors in the ϵ range. For our case, Minkowski distance, a general version of Euclidean and Manhattan distances, is used as the distance metric.
- *Directly density-reachable*: a sample that is within ϵ neighborhood of the core sample.
- *Density connected*: when two samples are directly density-reachable from a third sample.
- *Cluster*: a subset of samples where each sample is density connected.

By following these definitions, DBSCAN first identifies the core samples where clusters are going to be expanded upon. After finding a core point, all directly density-reachable points from the core is added to the current cluster. Later, all newly added samples are tested if they are also core. If so, their neighbors are also added to the cluster and this process is repeated until no samples left. Any point that is unreachable is considered to be noise. This approach tries to ensure maximum density reachability: dense clusters. Using this approach, DBSCAN is able to detect the clusters in the data, segments in our image, that

have arbitrary shapes and numbers. However, the parameters are in need to be manually adjusted.

5.3 HDBSCAN

HDBSCAN [27] is a clustering method based on the *connectivity* concept of DBSCAN. The main difference comes from its hierarchical cluster structure. HDBSCAN redefines the neighborhood relationship as follows to increase the stability of the resulting clusters, compared to the density-based methods like DBSCAN.

- *Core distance*: the distance to farthest neighbor within m_{pts} , which is a user-specified parameter.
- ϵ -*Core object*: a data point with core distance lower than an ϵ value.
- *Mutual reachability distance*: the maximum of core distances or direct distance between two data points.
- *Mutual reachability graph*: a graph where data points are connected with their mutual reachability distances.

The first step of the process is to determine the core distances for each point in the dataset. Then, construct a minimum spanning tree (MST) that will represent the mutual reachability graph (see Figure 5.1), which is based on mutual reachability distances between points. We construct the tree using reachability distances as weights.

In order to convert the MST into a hierarchical structure of clusters, the tree needs to be cut from its weakest connections. For this purpose, the weights are sorted in decreasing order and the highest weight is removed iteratively. A cluster can survive by just losing few points, can split into two whole clusters or disappear completely. A cluster disappears when it is left with fewer points than m_{clSize} . For convenience, m_{pts} is used in place of m_{clSize} .

After obtaining the hierarchical structure of clusters, the clusters need to be filtered based on their stability. The stability of the cluster depends on the density threshold, $\lambda = \frac{1}{\epsilon}$, which increases slowly towards deeper into the tree as higher weights (ϵ), are being eliminated. The λ values from cluster's birth to disappearance are taken into account as

$$S(C_i) = \sum_{x \in C_i} (\lambda_{max}(x, C_i) - \lambda_{min}(C_i)), \quad (5.3)$$

where C_i is the current cluster, *min* and *max* describe the λ values at the birth of the cluster and the removal of point x from it, respectively. The final tree takes the form shown in Figure 5.1. The “stable” clusters are now searched from bottom to top, by comparing the sum of stability of the children with their parent's stability. In case all of the children are more stable, the parent is skipped and vice versa. When the traversal reaches the root, the algorithm terminates with only the most stable clusters are remaining.

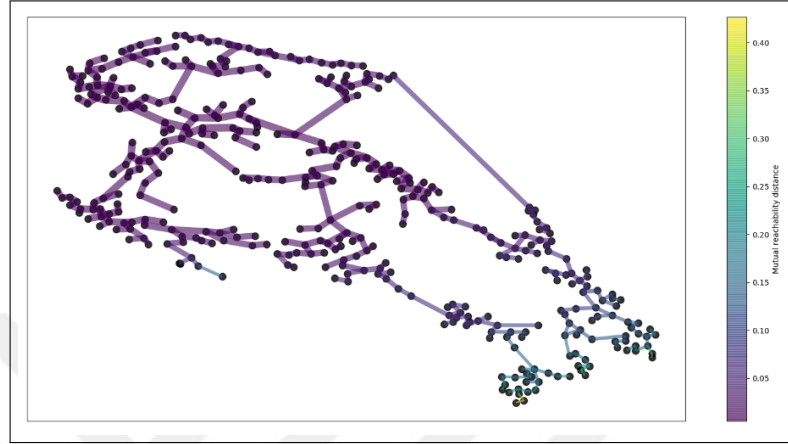
5.4 Results and Discussion

For segmentation, we took the first frame of the video as input. As all videos are colored, the images have multiple color channels, which is three in the following test cases. Therefore, every pixel in the image is represented with five features: two spatial and three colors. When testing, the image is converted into several color spaces: RGB, LAB, LUV, and HSV.

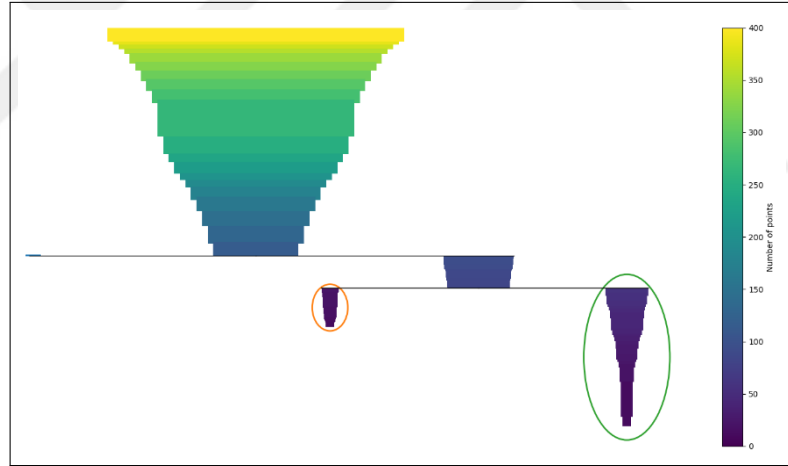
In order to improve clustering performance for images with hard shadows where context is hardly distinguishable, we implemented the shadow removal method from [75], [76]. Shadows are detected by thresholding the image in LAB color space per channel, then the pixels of each region are multiplied with a certain ratio. The ratio of inner and outer pixel color values, separately for each region, is used as the aforementioned ratio. An example can be seen in Figure 5.2.

For assessment, we took the first frames of our videos and segmented them with the three methods in various color spaces. The input pedestrian tracking data is used to determine the navigable regions in our frames. Example results can be seen in Table 5.2. The first two rows contain the original frame together with its pedestrian trajectories, which is the output of *Inception* (see Section 4.3.1). The segmentation was performed on a personal computer with Intel®Core™i7-4500U CPU @1.8 GHz, 8 GB RAM, and NVIDIA 740M. In terms of the computational speed, DBSCAN is the fastest, but both quantitative and qualitative results of other methods sometimes compensate for the lack of speed. Overall, HDBSCAN had the best performance in segmentation and navigable area extraction. The difference between DBSCAN and HDBSCAN is best visible in segmented outputs (colorful figures). DBSCAN seems to be more sensitive to noise, which makes it less robust. In Video-2, DBSCAN gets very confused at the upper left corner whereas HDBSCAN handles it much better.

We used both spatial and color features for each image when we ran the Mean-shift algorithm. Compared to other methods in Table 5.1, its resulting precision values for all videos are usually lower. Qualitative results are sometimes worse than DBSCAN based methods too. We tested multiple parameters for bandwidth determination when working with Meanshift. In the manner of speed, we obtained similar results with HDBSCAN.



(a)

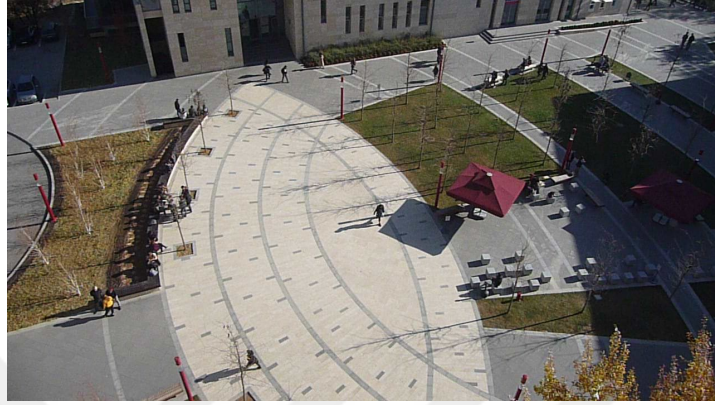


(b)

Figure 5.1: The resulting MST (a) and hierarchical structure (b) of a 20×20 gray scale image. Because the data is not 2D, sample positioning in (a) should not be taken into account. Therefore, the lower right cluster is actually sparser than others. In (b), selected clusters are shown in circles. Their stabilities are calculated to be higher than their parent's stability.

Table 5.1: The quantitative comparison of tested segmentation algorithms.

Video	PETS09-S2L1		Video1		Video2	
Method	Recall	Precision	Recall	Precision	Recall	Precision
DBSCAN	0.95	0.91	0.91	0.78	0.84	0.63
HDBSCAN	0.96	0.87	0.80	0.82	0.88	0.69
Meanshift	0.99	0.86	0.84	0.72	0.98	0.63



(a)



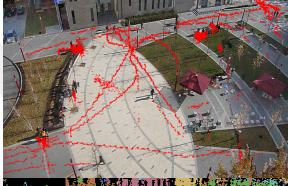
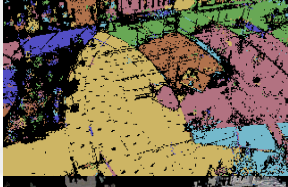
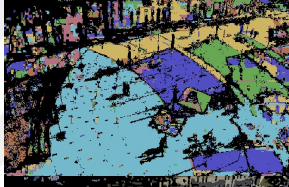
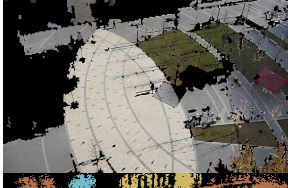
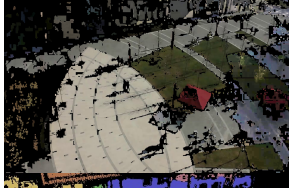
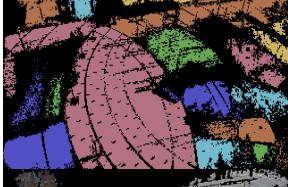
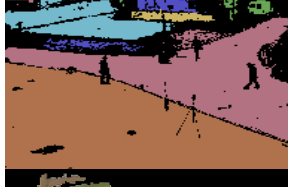
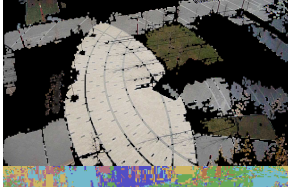
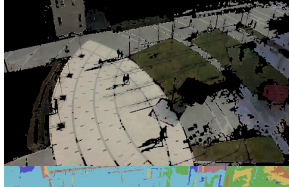
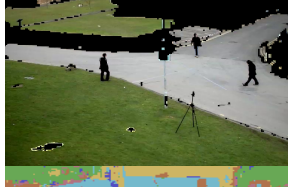
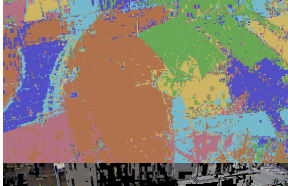
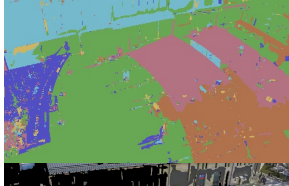
(b)



(c)

Figure 5.2: Shadow detection and removal process on an example frame. The majority of the detected regions in (b) are hard shadows in (a). As it is seen in (c), the removal process was successful for the region at right. There are some visible artifacts (overly colorful areas around some region borders), the clustering process eliminates them from the final result.

Table 5.2: Segmentation results including pedestrian trajectories, segments and navigable areas. Colors are only for visualization, does not always indicate identical clusters.

Video file	Video-1 (1280 × 720)	Video-2(1920 × 1080)	PETS09-S2L1 (768 × 576)
Original			
Trajectories			
DBSCAN			
HDBSCAN			
MeanShift			
			
			
			

Chapter 6

Navigation Area Reconstruction

After obtaining the segmented image of navigable regions from the source video, we will simulate artificial agents on them in a 3D environment. In order to reconstruct such a scene, we need to generate the 3D model of navigable regions and locate a camera in the scene which represents the original one in the video. For this purpose, we utilized computer vision techniques including RANSAC based horizon detection, homography based perspective correction and solution for the perspective-n-point problem. Our framework divides this process into several subsystems (see Figure 6.1):

- Find the vanishing points in the given frame using line cues from scene image and navigation data from the pedestrians. RANSAC is used for finding the most suitable horizon and nadir vanishing point definitions.
- Using the vanishing point information and pedestrian trajectories, perspective correction is applied to the segmented image; in order to obtain a bird-view perspective over the navigable regions.
- The bird-view image represents the blueprint for the 3D model that is going to be navigable for agents in the scene. For simplicity, we assume the navigable region doesn't include any changes in elevation, such as stairs. The navigable region isn't necessarily a single connected area.

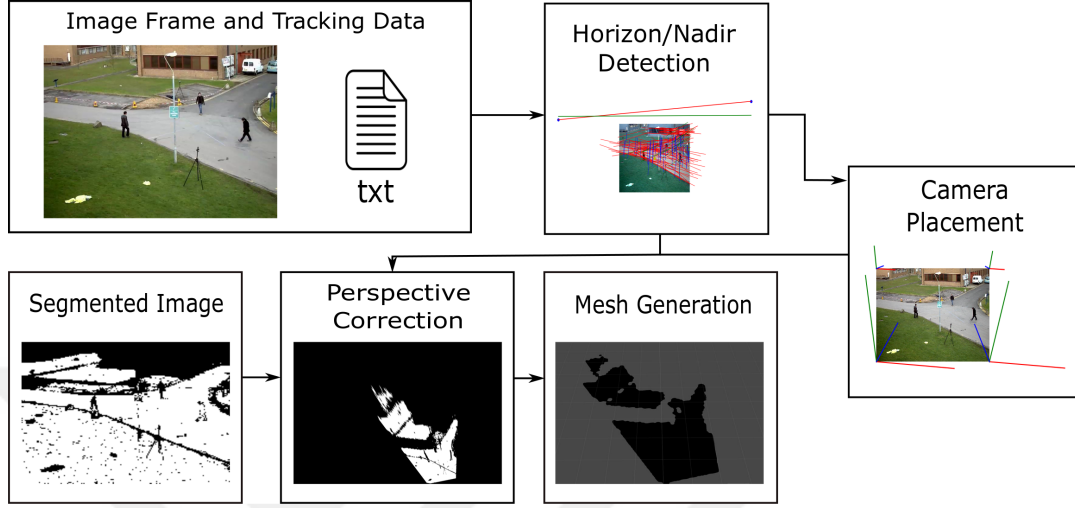


Figure 6.1: The overview of the navigable area reconstruction.

- To determine the initial placement of the camera, perspective-n-point solution based on Levenberg-Marquardt optimization is applied.
- As the initial placement is imperfect, because of distortions from perspective correction, the model is modified to obtain a perfect fit with the corresponding regions in the video. This is done by using homography to find the desired model endpoints that would correspond to the corners of the given frame, thus matching the navigable regions in it. After this step, perspective-n-point solution is applied again, to find the final camera placement.
- The 3D navigable mesh's location will be constant in Unity: lower left at $(0, 0, 0)$, therefore the orientation of the camera in the scene is determined relative to the mesh. For this purpose, the solution from the previous step is reversed accordingly.
- To be used in simulation, the camera placement and configuration information is written into a text file and read by Unity. Unity places the mesh to its default position and camera according to the text file.

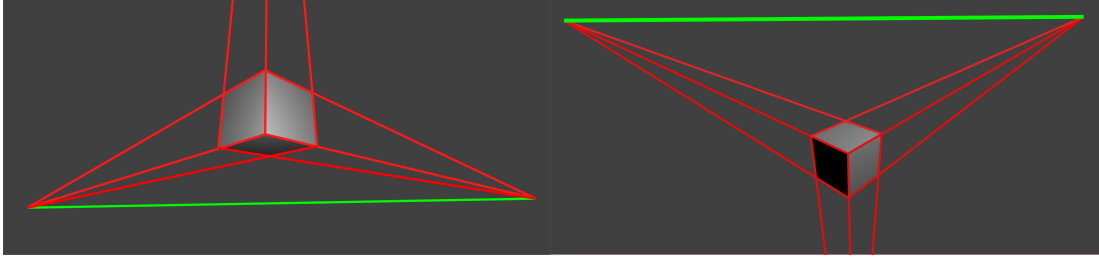


Figure 6.2: Difference in vanishing lines for different camera orientations. The image on the left shows the resulting horizon for the worm’s eye view, and the image on the right shows it for the bird’s eye view (in green). Because surveillance videos are shot in bird’s eye view, we expect our horizon to be above the center of the image.

6.1 Horizon Detection

The first step of our reconstruction framework is obtaining information of vanishing points in the scene. Vanishing points in an image represent the scene structure that is observed from a certain viewpoint. One can extract the orientation and internal calibration of the camera using the vanishing point locations in the image. Therefore, in order to place our camera in Unity, we need to first understand our scene in the sense of vanishing points’ orientation.

In a man-made scene, most of the lines are assumed to be parallel or orthogonal to each other in the real world. Each intersection of parallel lines determine the vanishing points in the scene. In our case, there are three vanishing points to be found: two that determines the horizon in the scene and a third one that is placed below or above the horizon. The side of the third VP depends on the camera’s placement in the scene; specifically its tilt angle. If the camera is placed such that it looks above, it is called worm’s eye view and its opposite is called bird’s eye view. Example illustration can be found in Figure 6.2. As our scenarios are based on surveillance systems that observe the navigable regions from a height, our camera configuration will be based on bird’s eye view; therefore we will expect our horizon above image center and third point below it. The third point will be referred to as “nadir” in the rest of this work.

Our framework is based on several approaches for finding vanishing points in

related work. The simplest way is to utilize lines from the scene [30]. Similarly, we utilize image lines by calculating their intersection. Parallel, horizontal lines in the image, meet at the horizon where vertical lines meet at nadir, as shown in Figure 6.2. These lines are extracted from the image using Hough transform where bilateral filtering applied as a pre-processing step to make edges sharper. Given points, i, j, u, v in the 2D image where every two points form a line parallel to each other, the vanishing point can be found using the properties of homogeneous coordinates. The cross product of each point yields a line, where cross product each line yields a point in homogeneous coordinate system. Therefore, crossing the given points and then the resulting lines result in the vanishing point (see Equation 6.4). It is divided by its last component, where points have the value of one initially, to normalize the result. The vanishing point is determined as

$$p_i = [x_i, y_i, 1], p_j = [x_j, y_j, 1], \quad (6.1)$$

$$l_{ij} = p_i \times p_j, \quad (6.2)$$

$$vp = l_{ij} \times l_{uv}, \quad (6.3)$$

$$vp = vp/vp_z. \quad (6.4)$$

In cases where the visual cues from the image background are not sufficient, works such as [34], [36], [35] and [33] treat the pedestrians tracking data as a subset of lines that can be used to find vanishing points. They treated the pedestrians as vertical poles that change position between frames, with an assumption that they do not change their stance much. The postures of the pedestrians are used as parallel vertical lines where they are combined at the merit vanishing point. The posture changes between frames are used to generate trajectories of pedestrians that are used as additional line features. Having the head and foot positions of the pedestrians at each frame at hand, using the method mentioned at Section 4.1, head and foot trajectories for each pedestrian are obtained for user-defined intervals. As the height of the pedestrian is assumed to be constant through the video, the aforementioned lines are taken to be parallel in the real world; therefore they meet at the horizon in the image. The usage of pedestrian postures is depicted in Figure 6.3.

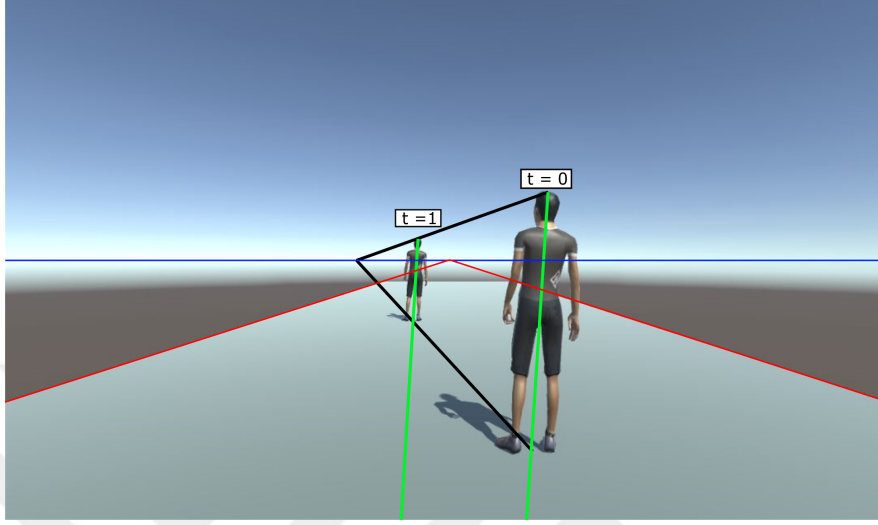


Figure 6.3: From the tracking data, the head and foot positions of the pedestrian at different times create parallel lines in the real world, which defines a single vanishing point. The red lines are extracted from the image and define another vanishing point where they together define the horizon, shown in blue. Additionally, the postures (head-foot combination for each pedestrian) is used to find the nadir, not shown due to being too far.

In our framework, we used the combinations of such line features and assessed their performance according to ground truth horizon and nadir vanishing point. As the pedestrian detection data and lines from the image contains noise, we use RANSAC [77] and thresholding to avoid them. We only considered a subset of the given lines depending on their length when we are applying RANSAC, as done in [39]. For every random line combination, a score is calculated according to the other lines in the subset. If the angle θ between the voting line and the potential vanishing point is below an empirically-determined threshold, the vanishing point obtains a score, as defined in Equation 6.5. The model with the highest count is considered to be the vanishing point. To generate ground truth for our videos, we provided a manual annotation tool to the user that is used to define parallel lines in the scene. The combination of these lines results in vanishing points of our ground truth.

$$vote(line_i, vp_{jk}) = \begin{cases} \frac{1-e^{\lambda \cos^2 \theta}}{1-e^{-\lambda}} & \theta \leq 5^\circ \\ 0 & otherwise \end{cases} \quad (6.5)$$

For camera calibration, we need to determine its intrinsic parameter matrix (K). It can be described as 6.6: where f is the focal length (same for f_x and f_y), we assume zero skew ($s = 0$), determine α as the aspect ratio and take the image center (c_x, c_y) at $(width/2, height/2)$. The focal length is calculated using the orthocenter of the triangle defined by the vanishing points. The orthocenter is used for calculation of the focal length as: $f^2 = |vp_1 - p||vp_2 - p| - |o - p|^2$ where o is the orthocenter, p is the projection of orthocenter on the horizon and vp 's are vanishing points on the horizon.

$$K = \begin{bmatrix} f & s & c_x \\ 0 & f \cdot \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}. \quad (6.6)$$

6.2 Perspective Correction

The vanishing point information obtained from the previous section is going to be used for correcting the perspective of our segmented image. For perspective correction, we projectively warp the image such that it is taken from a frontal view. This process is also called rectification and an example is shown at 6.4. In our case, we perform rectification to obtain a bird's view over the navigable region. When an image is taken from such angle, the view plane is parallel to the navigation area in the frame. In order to rectify an image, the concept of homography is used; as described in [62]. The 3×3 Homography matrix (H) determines the mapping between points in one plane to another up to a scale factor. Given points in one plane as x , the corresponding points in the second plane are calculated as $x' = Hx$. When working with homography, we define points in homogeneous coordinates.



Figure 6.4: Example perspective correction of Central Park in New York, USA. This correction is performed using four point correspondences between image planes, which are the corners of Central Park. The image on the left is courtesy of ASGG and Wordsearch [78].

In order to construct H , we need to have at least four point correspondences between two planes. The reason for four points is that the homography matrix is a projective planar transformation which has 8 Degrees-of-Freedom (DOF) and every point contributes two DOF. For more details, please consult Chapter 2 of [62]. In our scenarios, we do not have such point correspondences as it requires knowledge of the scene, such as a window with a known shape in the real world. Another way of constructing H , called *stratified rectification* ([37], [38]), is to look at its decomposition (see Equation 6.7). The leftmost matrix H_s is the similarity matrix, which contains rotation, translation and scaling components of homography. The similarity matrix is also called *the metric part*. H_a is the affine transformation matrix and H_p is the projective transformation matrix. Each matrix contributes 4, 2, 2 DOF, respectively.

$$H = H_s H_a H_p = \begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\beta & -\alpha/\beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & 1 \end{bmatrix}. \quad (6.7)$$

In our framework, we construct H starting from the projective transformation matrix. In Equation 6.7, the bottom line of H_p corresponds to the horizon of the image in homogeneous coordinates, $l_\infty = (l_1, l_2, 1)$. The homogeneous horizon is obtained with $vp_1 \times vp_2 = l_\infty$ where each vp is a vanishing point on horizon. By

applying the projection matrix to our image, we would affine rectify it where we recover parallelism of lines. Next step is to recover the metric properties of the image: such as length ratios and angles of non-parallel lines. This is crucial for our resulting model to be as accurate as possible. Affine transformation matrix H_a is used for metric rectification. The parameters α and β in H_a are calculated using the concept of circular points [79]. According to [37], there are three methods to determine the circular points:

- the known angle between lines,
- the equality of unknown angles, and
- the ratio of lengths.

Any of these that will be used must be known in the world plane. Each method defines circles with center $(c_\alpha, 0)$ and radius r , where first axis is α and second one is β , which is imaginary. The intersections of the circles determine our affine parameters α and β . In order to find those circular points, Bose and Grimson [40] utilized the trajectories of moving objects to be used for ratios of lengths in the image. They extracted the straight path segments of the tracked object and took the ones with constant speed. Then lines are used in 6.8 and 6.9 to calculate the centers and radius of each circle. Each line is defined as two points p_1 and p_2 , where s is the length ratio, $\Delta x = p_{1x} - p_{2x}$ and similarly for Δy . In our framework, we utilized the feet trajectories of pedestrians as non-parallel paths in the image. We assumed pedestrians have the same velocity in the real world, don't change their velocity substantially and took all paths with constant velocity. Therefore, s is taken as 1. As there are many intersection points for circles exist, we took their average. The resulting point (α, β) is then used in the affine transformation matrix.

$$(c_\alpha, c_\beta) = \left(\frac{\Delta x_1 \Delta y_1 - s^2 \Delta x_2 \Delta y_2}{\Delta y_1^2 - s^2 \Delta y_2^2}, 0 \right), \quad (6.8)$$

$$r = \left| \frac{s(\Delta x_2 \Delta y_1 - \Delta x_1 \Delta y_2)}{\Delta y_1^2 - s^2 \Delta y_2^2} \right|. \quad (6.9)$$

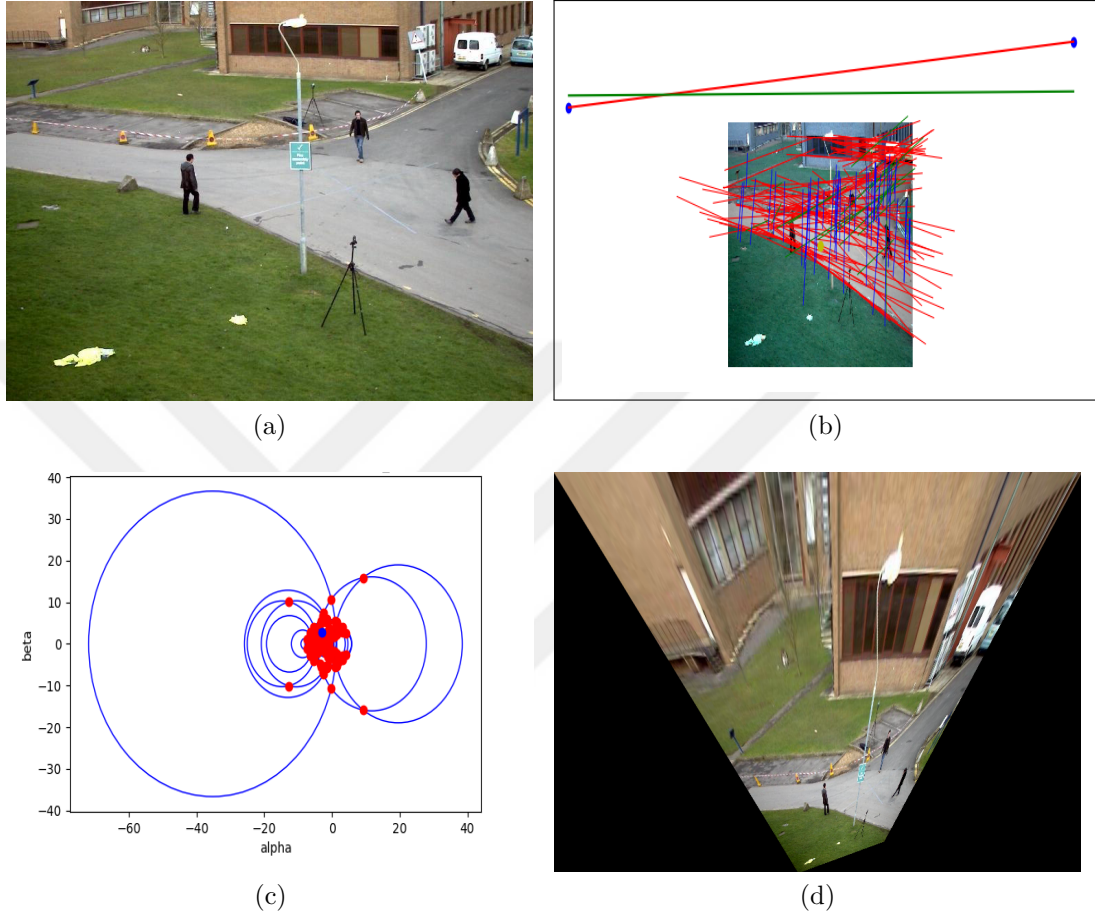


Figure 6.5: The overview of the metric rectification process for PETS09-S2L1. After determining the horizon for the image to calculate the projective transformation (b), the pedestrian trajectories are used to find the circular points to be used in affine transformation (c). The resulting homography matrix is applied to warp the image as if it is taken from a bird's eye view (d).

We apply similarity transformations to our resulting metric-rectified image, as described in [39] to keep its features within the image boundaries. Figure 6.5 illustrates the stages of the metric rectification process. The initial rectification has errors from projection, but the camera placement process will refine it so that it is closer to a frontal image.

6.3 Camera Placement

After applying metric rectification on our segmented image to obtain a bird-view perspective, we are going to determine the orientation of our camera so that the navigable model’s projection on the view plane matches its real-world counterpart. The projection in our videos is based on pinpoint camera model. Such model projects 3D points (X, Y, Z) in the world scene to 2D image points (u, v) based on its projection matrix, realized in Equation 6.12 as P [62]. The matrix K is called the intrinsic matrix of the camera, which contains the focal length information and image center. The extrinsic part of the camera model, $[R|t]$ in Equation 6.11, includes transformations that define the orientation of the camera in world coordinate space. It is also called *the joint rotation-translation matrix*. In this section, we describe how we approximate the extrinsic part of the camera model, as we already constructed intrinsic matrix using the vanishing points (see Section 6.1).

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = P \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \quad (6.10)$$

$$P = K[R|t], \quad (6.11)$$

$$P = \begin{pmatrix} f & 0 & c_x \\ 0 & f \cdot \alpha & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}. \quad (6.12)$$

In order to find the transformation expressed in the extrinsic matrix, we utilized perspective-n-point (PNP) solutions that use point correspondences between 2D image points and 3D world points to approximate camera orientation. A stable implementation of an iterative PNP solver was provided by OpenCV library [56]. The iterative method uses Levenberg-Marquardt optimization [80] to minimize the re-projection errors through finding the optimal orientation for the camera.

This method is able to refine its camera pose solution if it is provided with an initial guess in the form of Rodrigues rotation matrix and translation vector.

The warped segmented image resulted from the previous section is going to be used as the blueprint of our 3D navigable area model. Therefore we are going to use its four corners as points in our 3D world frame that correspond to corners of our image's corners in the image frame. OpenCV's coordinate system is right-handed with Y axis down where Unity's is left handed and Y is up. As the model is going to be placed in Unity, we adjust the model corner coordinates accordingly. The model in Unity is 2D and placed on the $X - Z$ plane, therefore Z axis of model coordinates are taken as $image_{height} - v$ from the image and Y is 0. When running the PNP solver, we provided the intrinsic matrix K and assumed zero distortions that would affect the projection. The initial run of the solver is expected to be inaccurate because of metric rectification and focal length errors. An example output can be seen in Figure 6.6 (a). The axes that represent the projections of the four corners of the model are away from the corners of the image (see Figure 6.6 (b)). We would take pose estimation as successful if the model's projection to match image corners with very low error (2-3 pixel difference).

Assume that there is such a projection matrix P that maps the 3D corners of the model to their projections on the image plane (Equation 6.13). This projection matrix is the inverse of one that would map the original image corners in the 2D image plane to the ideal model corners; as the camera is identical for both cases. However, the inverse projection of an image point is ambiguous as there are infinite points that are projected on it. To solve this problem, as we assume our model is 2D and sit on the $X - Z$ plane, we remove Y component of the 3D position vector. This turns our problem into a plane to plane homography. The PNP solver is used to obtain the initial model corners' projection result on the image plane. $H_{projection}$ is found between model corners on the model plane and projection results on the image plane using 4 point correspondence (Equation 6.14). Then by applying $H_{projection}^{-1}$ to the original image corners, we would obtain the corners for the corrected model on X-Z plane (Equation 6.15). Because we know the perfect model, we use homography to warp our existing model corners to it (Equation 6.16). The homography matrix $H_{adjustment}$ is found

using 4 points correspondence between each models' corners on X-Z plane. After adjusting the model, camera pose is estimated again using iterative PNP. To keep the adjusted model's corners inside the image to not leave any region outside, the initial model is minimized and similarity transformations are applied to the final model. Figure 6.6 (c) and (d) summarize this correction process. The result of PNP, together with image frame properties (e.g., resolution, center) is used in Unity. The field of view (FOV), found from the focal length, is used in Unity as the vertical FOV.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{projection} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{initial} \quad (6.13)$$

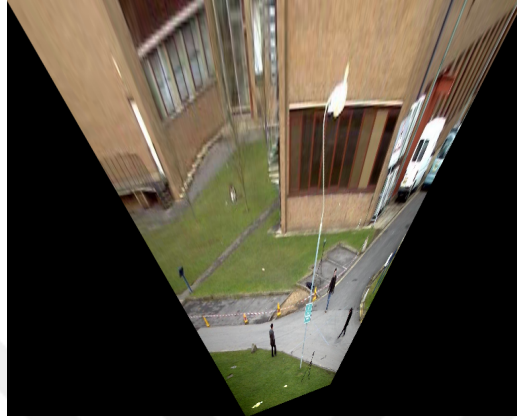
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{projection} = H_{projection} \begin{bmatrix} X \\ Z \\ 1 \end{bmatrix}_{initial} \quad (6.14)$$

$$H_{projection}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{corners} = \begin{bmatrix} X \\ Z \\ 1 \end{bmatrix}_{ideal} \quad (6.15)$$

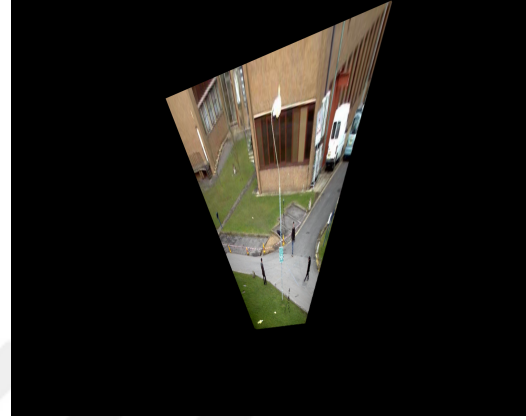
$$\begin{bmatrix} X \\ 0 \\ Z \\ 1 \end{bmatrix}_{ideal} = H_{adjustment} \begin{bmatrix} X \\ 0 \\ Z \\ 1 \end{bmatrix}_{initial} \quad (6.16)$$

6.4 Navigation Mesh Construction

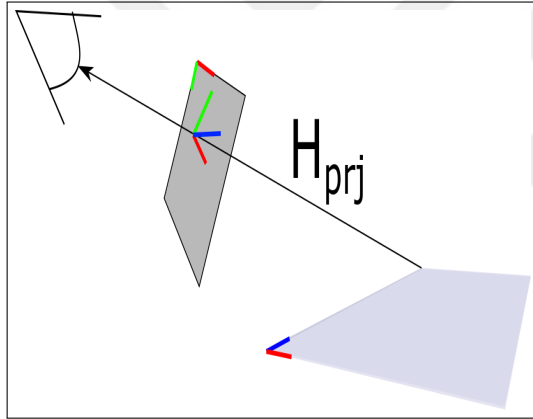
After determining the navigable regions in the video frame, they will be used for reconstruction of the 3d scene in which agents are going to be simulated. For this purpose, we implemented a mesh generator framework which takes a black-white



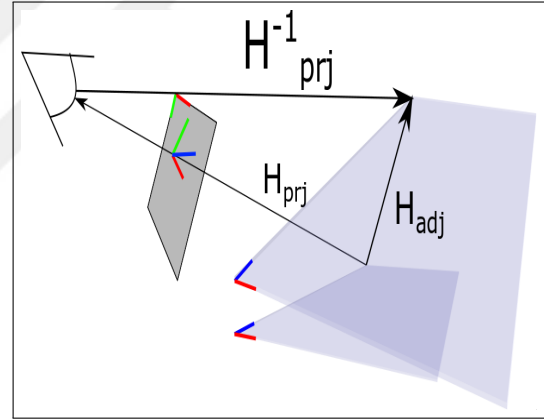
(a)



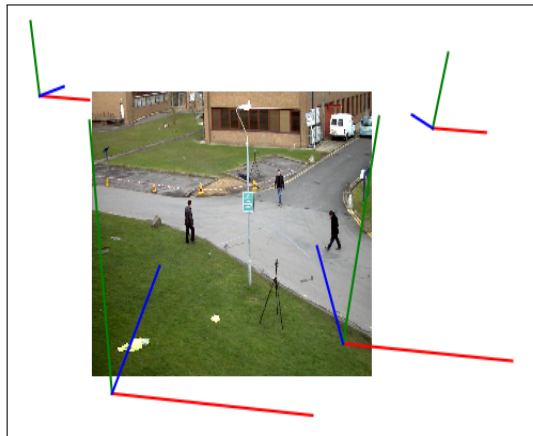
(b)



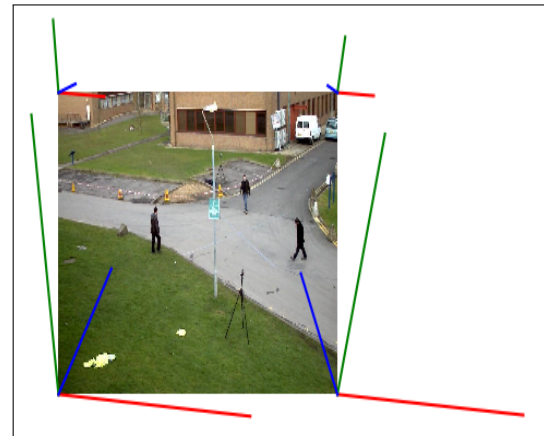
(c)



(d)



(e)



(f)

Figure 6.6: The model adjustment process starts with an initial solution to plane to plane homography using PNP solution (c) for (a). As the resulting placement (e) is noisy, H_{prj}^{-1} is multiplied with image corners (d) to obtain the perfect fitting model (b). In (e) and (f), RGB lines correspond to X, Y, Z coordinates in Unity. The model images in (a) and (b) are not in scale with illustrations in (c) and (d).

image and turns it into a 2D mesh. The navigable areas resulted from the process in 5 are converted into a black-white image to be used in this framework, where the white areas represent navigable regions. For simplicity, we assume that the area is flat: there are no stairs or any kind of altitude change in the environment.

As the image from segmentation framework can be noisy because of tiny clusters and holes, dilation and erosion operations are applied. Dilation operation enlarges the white areas and removes holes where erosion eliminates tiny clusters. After clearing the image, contours are found using OpenCV [64] functions. Contours contain information on corners, lines in the image. We used OpenCV’s tree hierarchy when obtaining the contours in the image. Such hierarchy numbers each contour from outer to inner, where outermost contours are considered to be at level 0 where children get higher numbers.

In order to convert this contour hierarchy into a mesh, we used *Triangle* [81] framework; which generates a mesh by triangulating the area defined by its border vertices and edges. *Triangle* takes input vertices and edges in the format of a PSLG (Planar Straight Line Graph). Every corner and edge in contours are directly transferred to *Triangle* in this form, similar to a (.poly) file. *Triangle* then applies Delaunay Triangulation to triangulate the region inside of PSLG. An example triangulation result can be seen in Figure 6.7. According to the given flags, *Triangle* applies different triangulations to input PSLG. In Figure 6.7, constrained conforming Delaunay triangulation is applied on PSLG. In this form, the edges from PSLG are included in the triangulation but not all triangles need to have the Delaunay property. For more details, the reader is referred to *Triangle* documentation.

It is noticeable that the triangulation in Figure 6.7 contains no holes; islands inside the triangulated area. In most cases, the mesh contains holes; they need to be determined using the contour information. A “hole” in a contour is understandable by its level and parent. If the contour’s current level is even (considering outer borders as 0), then it encapsulates a hole as the area between it and its parent is inside the mesh. Of course, a hole can appear twice while traversing the children, which gives us a disconnected mesh. This process is repeated for all

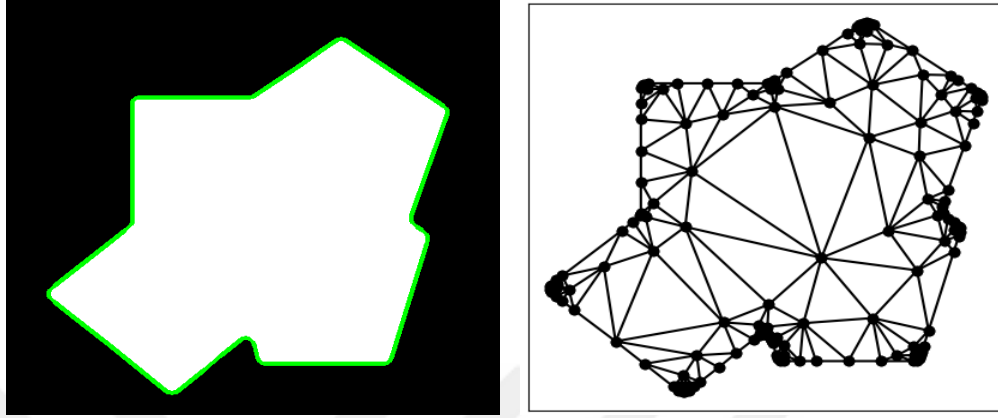


Figure 6.7: An example triangulation of a black and white image. The contours of the image can be seen in green on left. The right triangulation is generated using the “-q” flag in *Triangle*, which results in a constrained conforming Delaunay triangulation. Steiner points, extra points on borders, are added, which enable the related triangles to have the Delaunay property.

contours.

When processing holes in the mesh, *Triangle* uses the notion of “triangle-eating virus”, which removes the triangulation from its initial point to closest segment it reaches. The planting location of the virus needs to be within the whole area. In order to find such a point, we construct the hole as a polygon using its encapsulating contours and the contours it encapsulates. Then, we find a representing point inside the hole region using Shapely [82]. The representing points are sure to be inside of the polygon and act as the initial point of the hole virus. The mesh generated from the perspective correction can be found in Figure 6.8. After the triangulation, the resulting mesh is exported to Unity.

6.5 Results in Unity

Figure 6.9 shows our results for horizon and nadir compared to our ground truth calculations. Overall, we are able to detect horizon in an accurate manner even under complex scenarios like our custom videos. However, the postures of pedestrians seem to result in highly different focal length values than ground truth,



Figure 6.8: Example triangulation of the navigable area. The model image is preprocessed with erosion and dilation in order to refine it.

seen in Table 6.1. Even though focal length has no effect on perspective correction and its importance comes up when we are placing our camera. The camera placement, based on point-n perspective, considers the camera calibration when determining its orientation and we adjust our model accordingly. A shorter focal length results in a camera much closer to the navigable area than in real-world. This causes posture distortions when it is used in Unity, as shown in figure 6.12. Therefore, getting a good focal length estimation as important as getting a decent horizon orientation. The metric rectification process for each video is shown in Figure 6.10. A higher projection error results in a more significant change in each of our model. The generated models are also compared to the ground truth models, obtained via Google Maps [83] for PETS and Yandex Maps [84] for our custom videos. The resulting placement of our model in Unity together with dummy pedestrians are shown in Figure 6.11. Overall, we are able to fit the corrected navigable area model onto our input video with minimal error (1-2 pixel difference).

Table 6.1: The focal length values for test videos compared to the ground truth calculated from the annotated image cues.

Video	Focal Length	
	Obtained	Ground Truth
PETS09-S2L1	914.64	1262.94
Video - 1	2865.22	2751.36
Video - 2	2011.18	2197.55

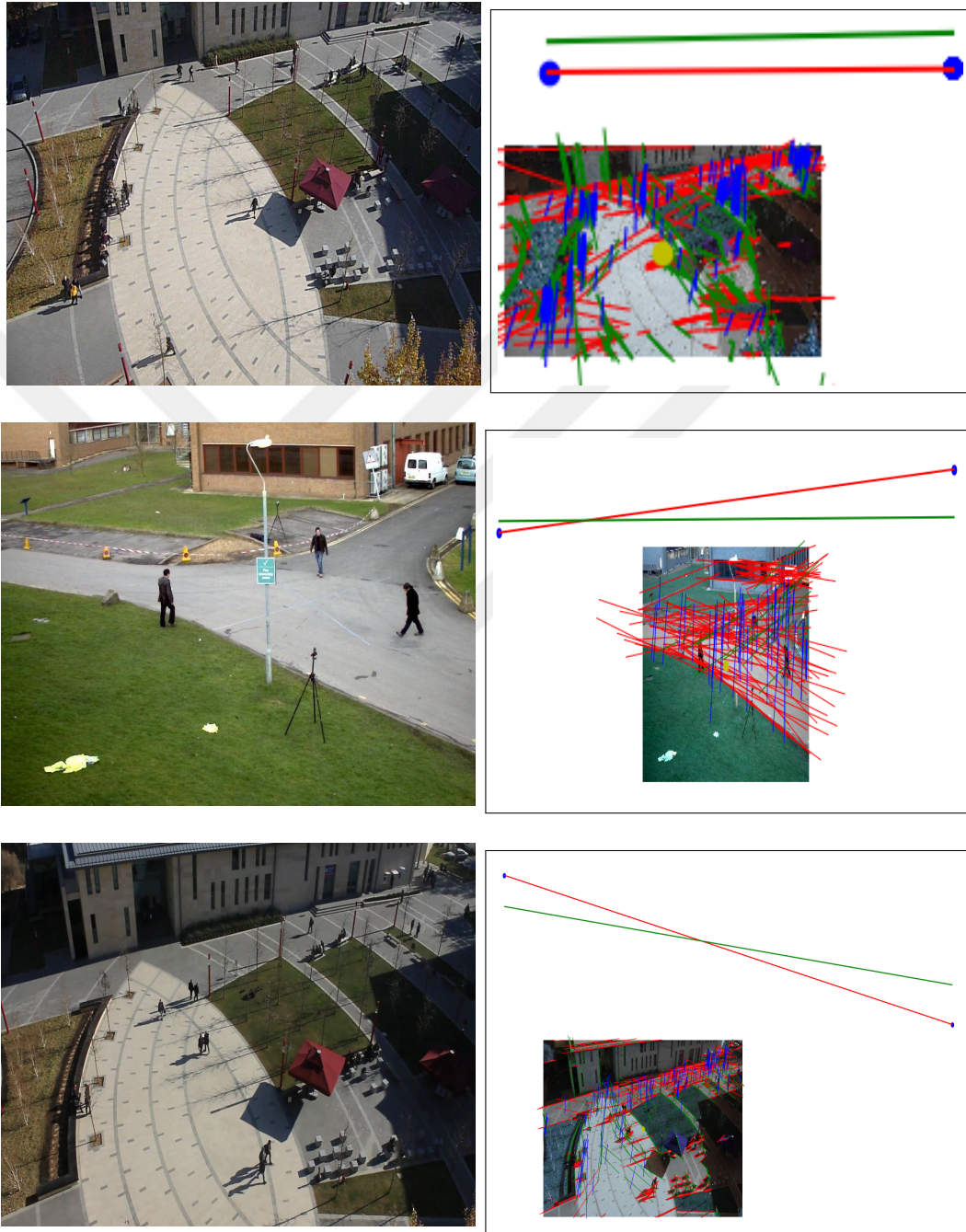


Figure 6.9: Example horizons calculated using the line features and pedestrian posture trajectories in the image. Blue points indicate best locations for each vanishing point. The green line is the ground truth, which is calculated via manual annotation. The nadir point is not shown, because it is too low in the image. The lines on the image represent the inlines for each vanishing point. The best results are obtained in PETS video, as the pedestrian postures are observed easily and the camera's angle is less complex.

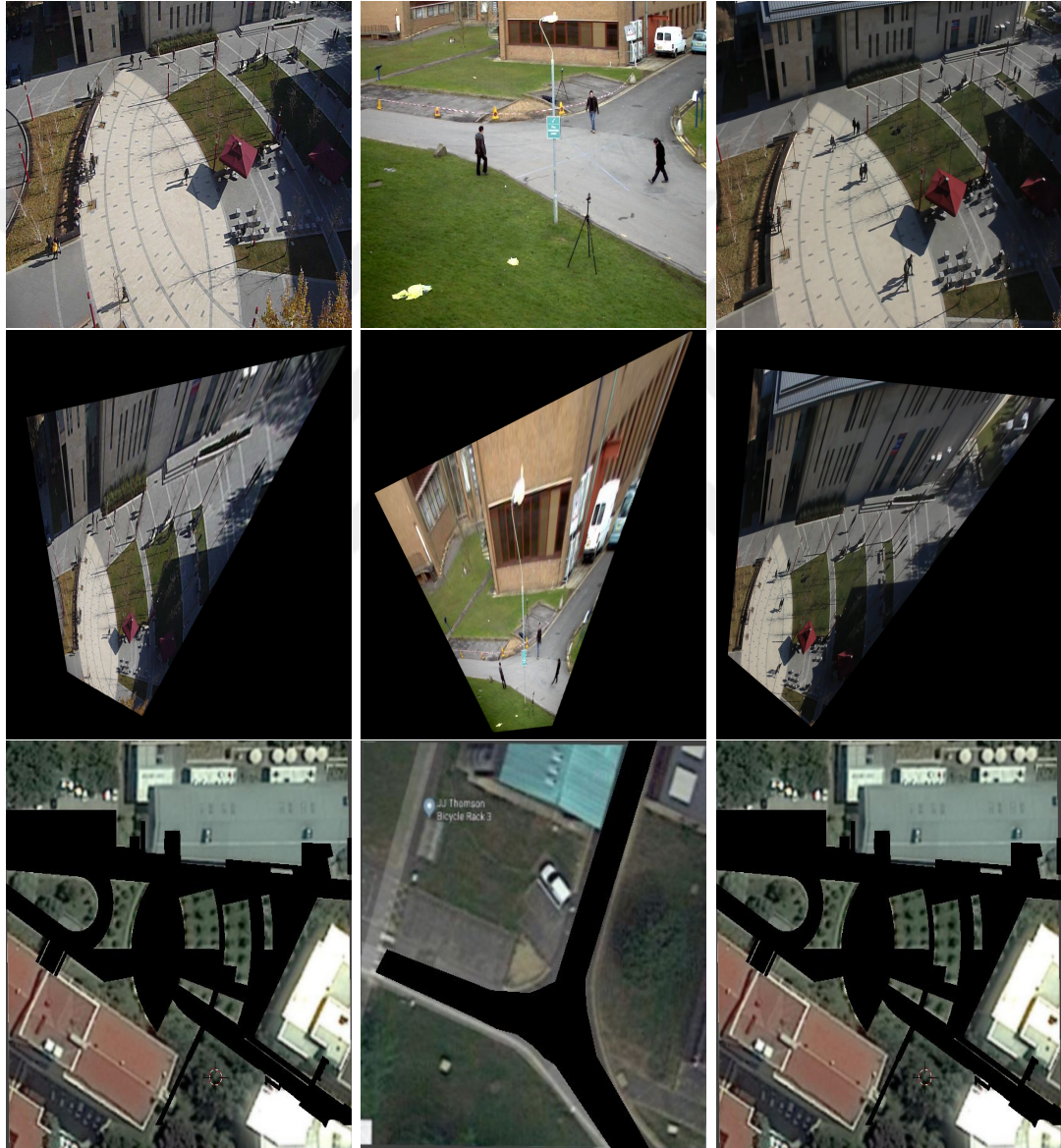


Figure 6.10: Each video together with their perspective corrected versions. The bottom row includes ground truth rectifications of navigable areas taken from online maps.

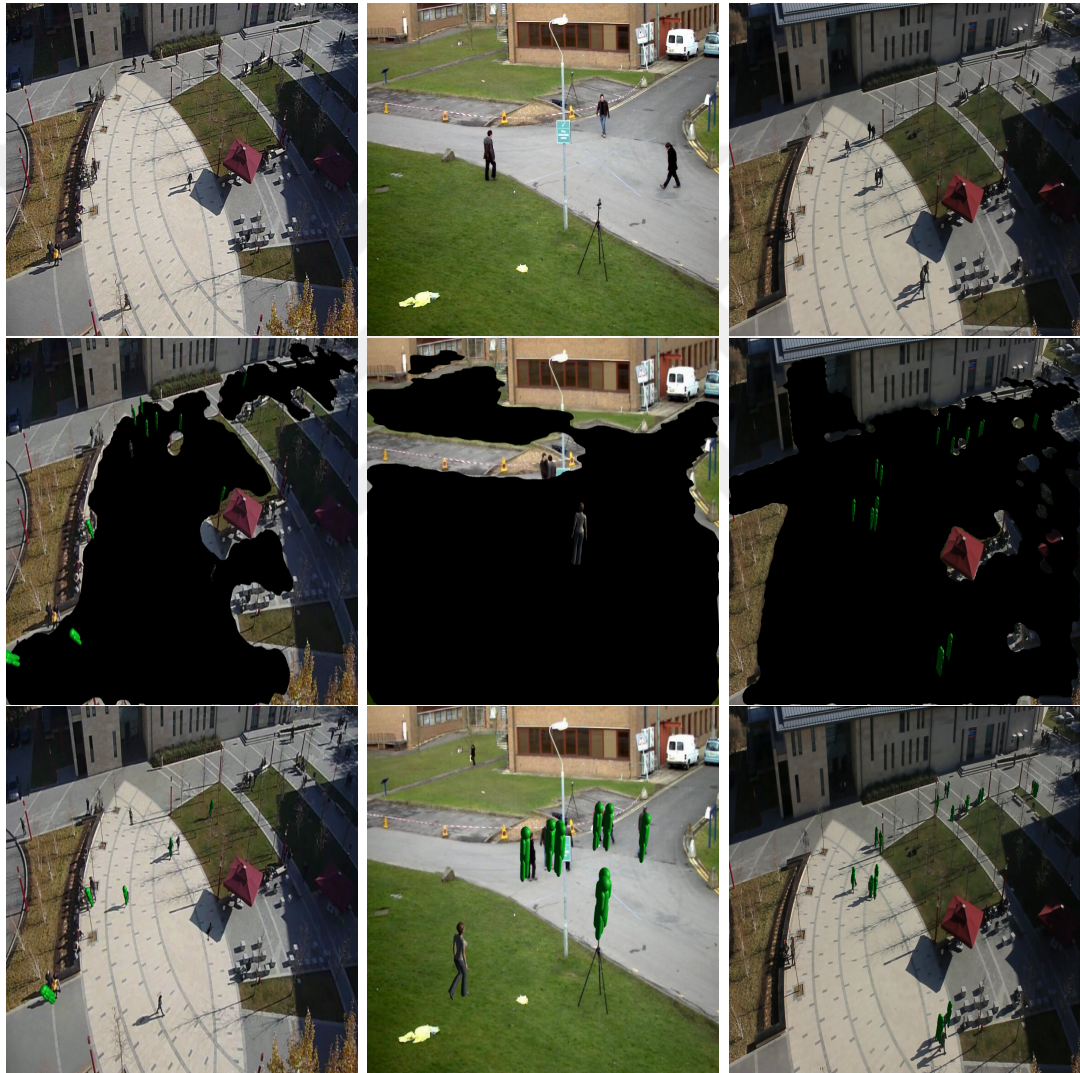


Figure 6.11: The placement of navigable area model for each video. The first row includes original frame, the second one includes model with dummies on it, and the third one is the final result in which the model is invisible.



Figure 6.12: Differences in postures of dummy models with pedestrians in the video. A camera with a focal length smaller than expected will be placed closer to the area, resulting with the posture in the left figure. Better postures are obtained when the focal length approximation is more accurate.

Chapter 7

Crowd Simulation and User Interface

In order to simulate our artificial and projected agents in a realistic way, we implemented pathfinding and collision avoidance techniques in the literature. For pathfinding, we placed the model in Unity 3D where its lower left is at $(0, 0, 0)$ and generated a navigation mesh in Unity according to the default pedestrian model size. The navigation mesh is used by agents to determine their paths that reach their goals on the 2D plane. Unity uses A* pathfinding algorithm [85] and Unity's navigation mesh takes into account clearance.

For collision avoidance, we used Reciprocal Velocity Obstacles (RVO) [86], [87]. RVO is based on velocity obstacles that are calculated for each pedestrian and used to determine the optimal movement vector that follows the path while avoiding collisions with other agents, if possible. In RVO, both agents are responsible for adjusting their velocity, in order to prevent oscillations [86], therefore direction change is mutual. However, the projected agents follow their own paths from the video without actively participating in collision avoidance. Therefore, full responsibility is on artificial agents. This might cause problems in case of a dense crowd as the artificial agents might fail to find an optimal solution and the unresponsiveness of projected agents might degrade the realism of the simulation.

The agents are projected on the navigable model by sending rays from the camera through the feet location of the tracked pedestrian on the image plane. The projection is refreshed at every frame and state of each projected agent is updated. In case a projection gets out of sync with the associated pedestrian's tracking information, means tracker lost the pedestrian, it is destroyed after a few frames. The projected agents take their latest displacement as their current movement vector, which is used in both simulation environment and RVO.

The height of all agents in the simulation is determined by the average height of the detection boxes in the 3D scene. We send another ray from the camera, r , this time to head position of the pedestrian in the image plane. Given the location of the camera in 3D as C and position of agent's feet as F , the head location H is determined in Equation 7.2. D in Equation 7.1 is the vector between the camera and agent's feet location on the XZ plane. In Equation 7.2, the distance from the camera to head in terms of ray unit vector $\hat{r}$ is found using D and added to the camera location. The height is relative to the unit distance in Unity 3D. We calculate the height at the beginning of the simulation for an initial approximation, but it can be specified by the user while the simulation is running, which takes into account the currently visible pedestrian bounding boxes on the screen. Default RVO parameters for artificial agents are determined according to their initial height.

$$D = (F_x, 0, F_z) - (C_x, 0, C_z), \quad (7.1)$$

$$H = \hat{r} \frac{\|D\|}{\hat{r} \cdot D} + C. \quad (7.2)$$

We provide an extensive user interface for users to easily augment the artificial agents, as shown in Figure 7.1. The program starts by taking the stabilized video to be augmented, the tracking data for projection, the camera calibration data and the navigable area as input. After the camera and model placement is complete, the simulation starts by projecting the video as background in camera frustum, where only the navigable area and agents are visible. After the simulation starts, the user has options to

- pause, resume and stop the simulation,
- load a background video, save the current view of the camera as video (using [88]),
- recalculate the height of pedestrians,
- load a new mesh to be placed, toggle the mesh's visibility,
- adjust the position and orientation of the camera and its focal length,
- toggle the visibility of artificial and projected agents,
- display the number of collisions between artificial and projected agents,
- add, remove, select (single or multiple) and control the artificial agents by determining goals for each.

In addition to options above, the user can also adjust the RVO parameters, by changing their coefficients for default values (except number of neighbors), of all selected artificial agents such as

- *number of neighbors* to consider while calculating velocity obstacles,
- *maximum speed* of the agent,
- *range* of the circular area where other agents are considered
- *reaction speed*, which is the minimum amount of time that the calculated velocity of the agent is safe with respect to the other agents

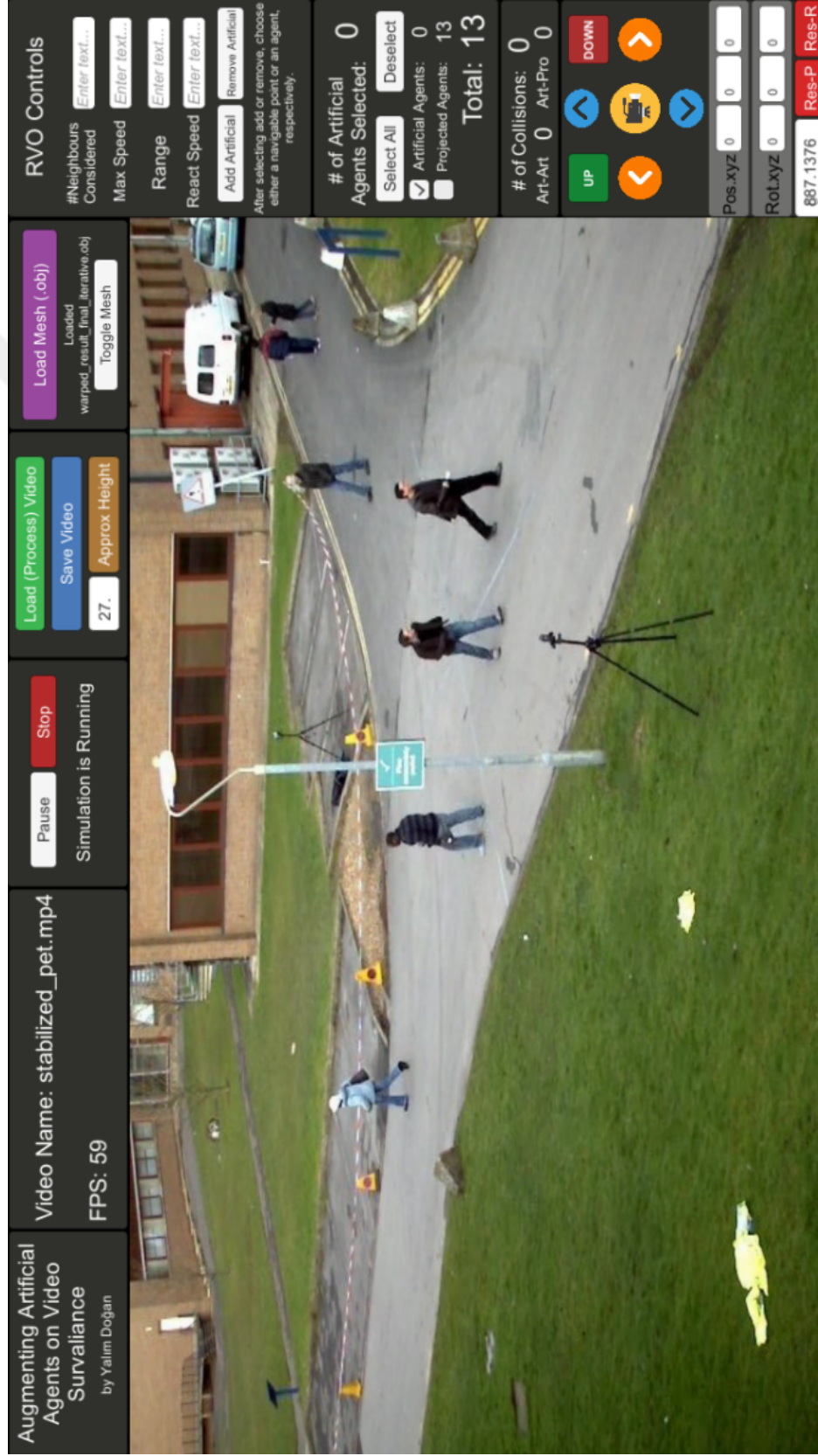


Figure 7.1: The graphical user interface of the framework.

Chapter 8

Results and Discussion

We presented a framework for augmenting artificial agents over the navigable regions in the given surveillance video. We have implemented our framework using Python 3 for computer vision subsystems and Unity 3D (2017) for simulation. We tested our framework with some sample scenarios. Figure 8.3 demonstrates example detections from each video together with their projections in Unity. In some cases, there are more projections than the existing detections because the projected agents follow their recent velocities for a few frames if no associated detection can be found. The projections mostly align, i.e., they have the same position and orientation, with the detection boxes they are associated.

Figures 8.1, 8.4, and 8.7 provide still frames from each video. Figure 8.2 contains an artificial agent controlled by the user (in yellow), which is redirected to the upper right portion of the navigable area. On its way, the artificial agent changes its path according to the velocity of the invisible projected agent, the person on left, thanks to RVO. A similar result in Figure 8.5 contains multiple projected agents and a manually-controlled artificial agent. The artificial agent first corrects its path towards its goal by avoiding collision. Then it also avoids collision with the other two agents on the right. In Figure 8.8, a projection disappears while our agent were avoiding it. This does not cause a major problem in this case because the projection is recovered after a few frames. When

multiple agents are simulated, the groups tends to separate because of the projected agents, as seen in Figure 8.6. Some projected agents approach aggressively towards artificial agents because they are not aware of them. This causes our artificial agent to be dragged, which is a unrealistic behavior. In Figure 8.6 (c), the bottom agent is left behind because the projected agent on the left dragged it for a few frames. In Figure 8.9, the dragging enforces both groups to form a line. However, artificial agents were able to avoid each other without a problem because they take mutual responsibility when changing velocities.

Compared to the other works in the literature on crowd pedestrian detection, we use a more sophisticated detection tool, which is based on convolutional neural networks. We automatically extract the navigable areas from the video and replicate them in usable 3D environments for crowd simulations. In addition, we provide tools for manipulating artificial agents in the simulation via RVO parameters and point-click goal definition. The reconstruction subsystem is able to construct the scene accurate enough where the artificial agents do not stand out because of their abnormal posture and height. Our framework has some limitations and potential areas for improvement.

- Our simulation only works on static, monocular cameras in which the area is viewed from above. It can handle cases with a pedestrian-eye view, as long as the navigable region is visible.
- Our navigable area extraction methods only consider trajectories of the pedestrians in the video, which might limit the area of navigable regions. This might cause invisible walls in the environment where artificial agents have a hard time finding an optimal path.
- The interactions between artificial and projected agents is limited to collision avoidance. Event detection systems could be added to the framework so that the behavior of artificial agents are more realistic.
- The behavior of virtual agents could be improved using the approaches described in literature, e.g., [89].

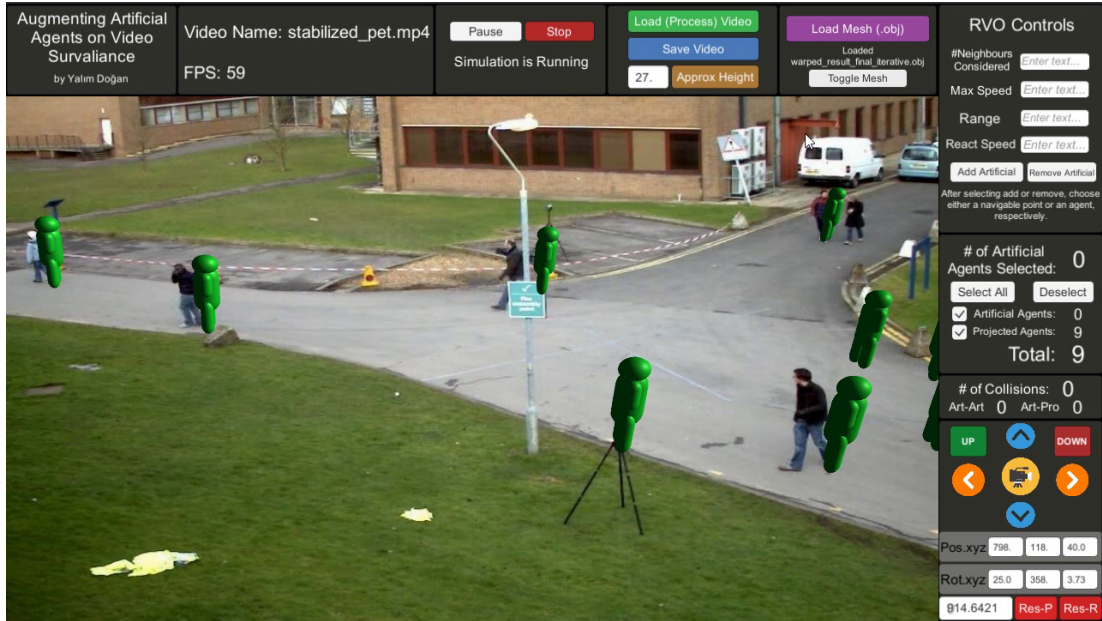
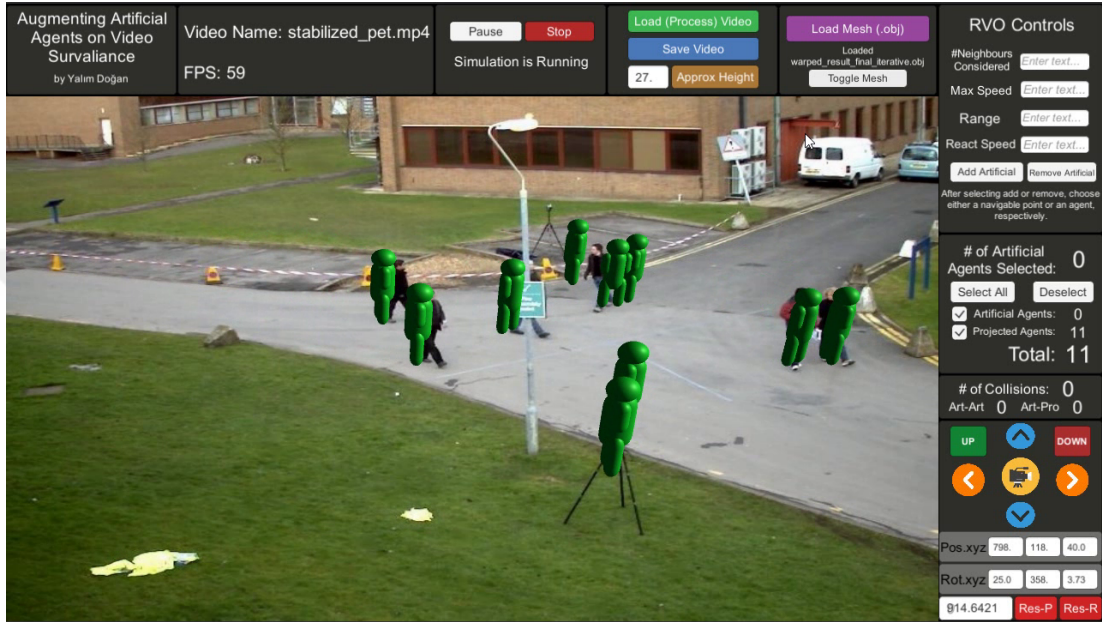


Figure 8.1: Screenshots from a simulation on PETS video. In cases where some projections stay behind the associated detection in the simulation, their high velocity would prevent the artificial agents from crossing their current detection box area.



Figure 8.2: Sample interaction between an artificial agent and a detection in video. As the projected agents are unresponsive, the artificial agents are expected to take full responsibility for collision avoidance. In normal cases, both agents mutually change their velocities.

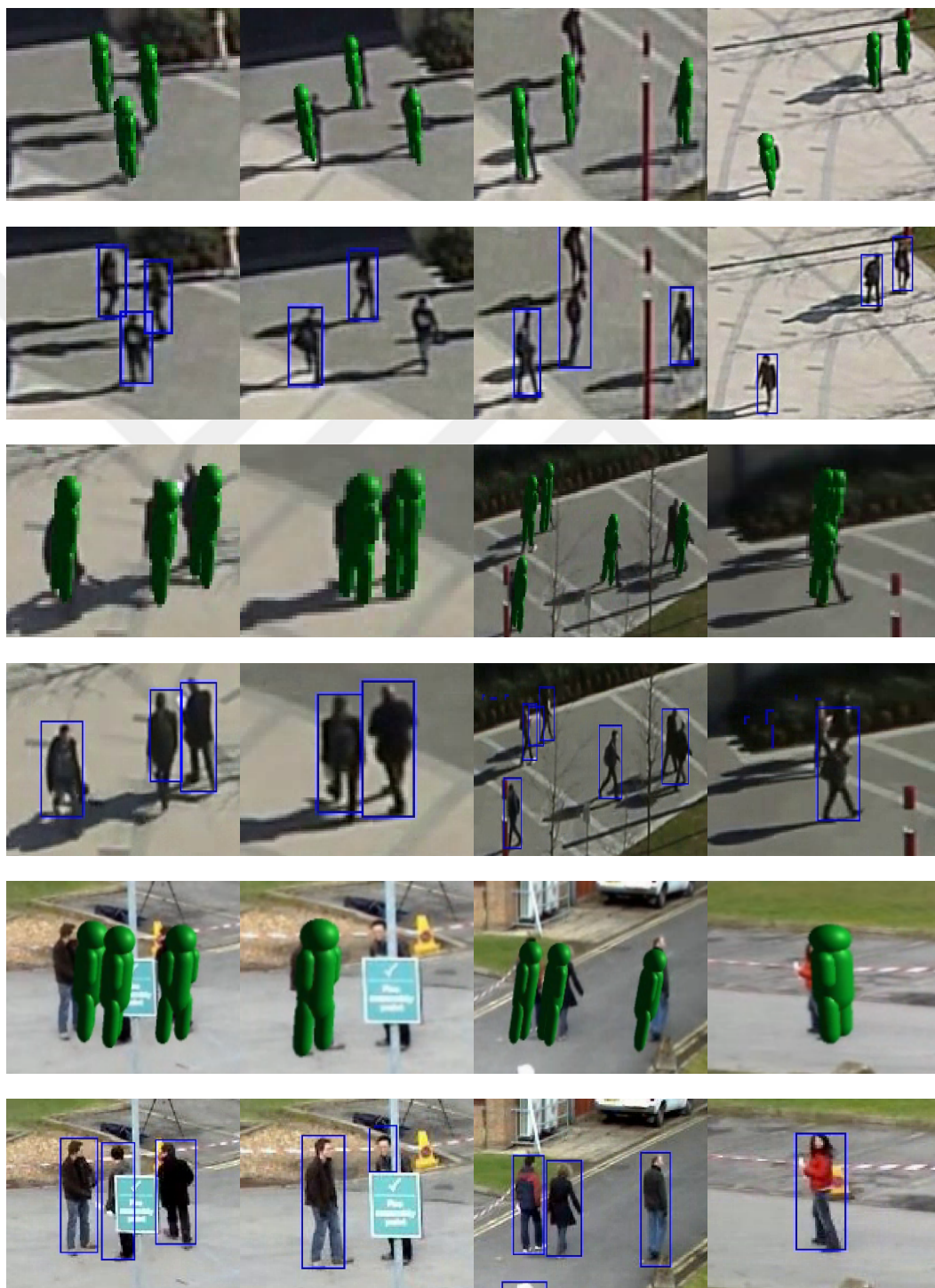


Figure 8.3: Sample detection and projection results for Video-1, Video-2, and PETS, from top to bottom, respectively.

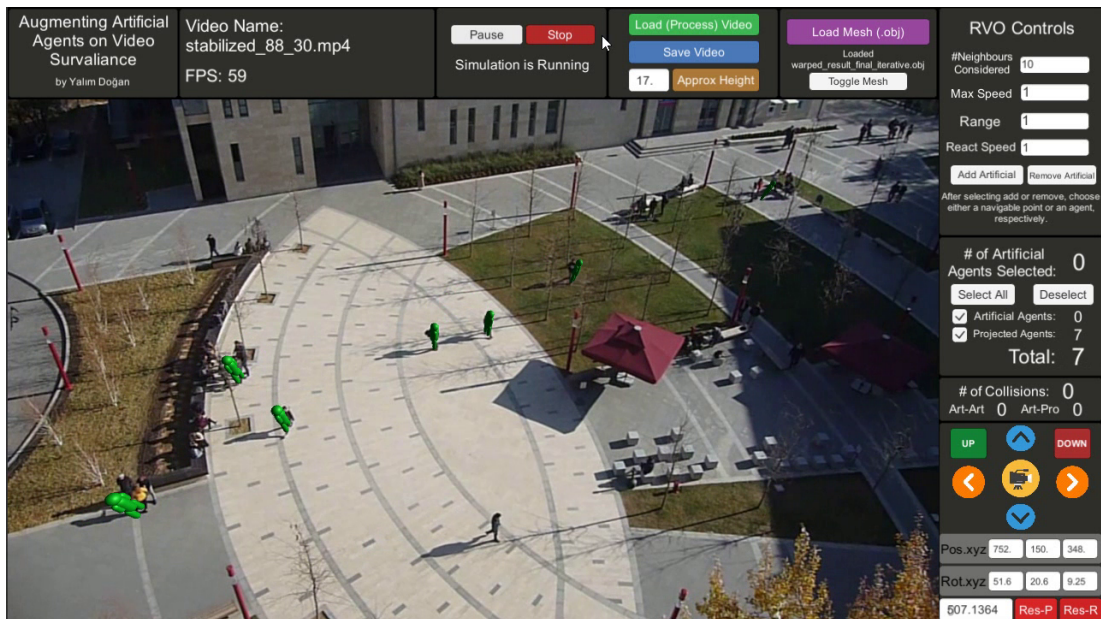
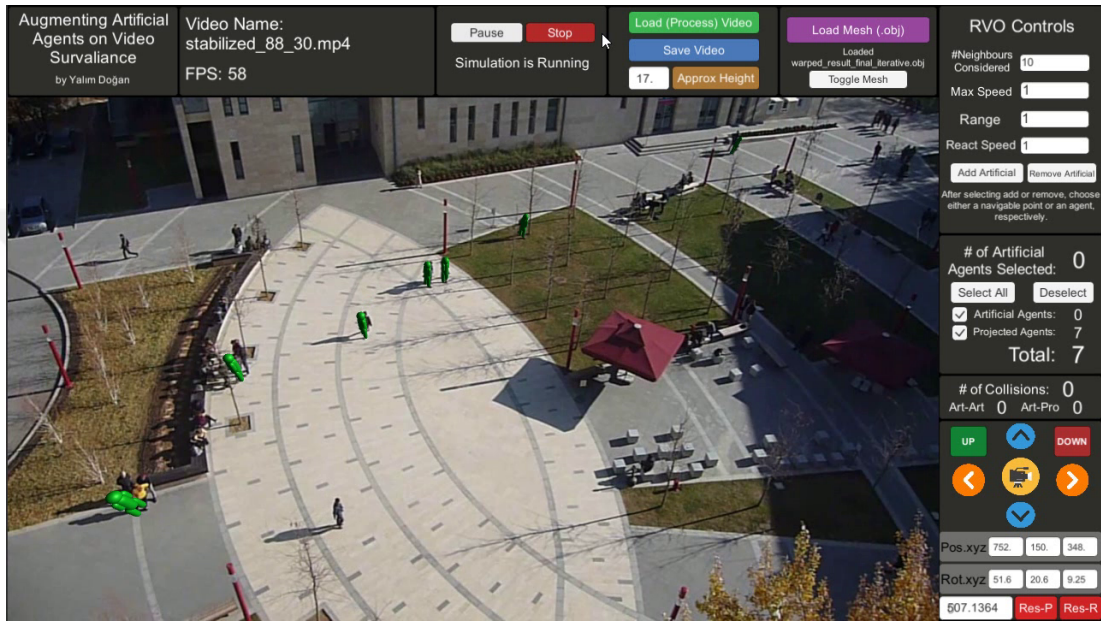


Figure 8.4: Sample projections from Video-1. Because the navigable area at top left is not recovered, the detected agents weren't projected.



(a)

(b)



(c)

(d)

Figure 8.5: A selected agent avoids collision with a projected agent and adjusts its path before preparing to avoid a couple in Video-1.

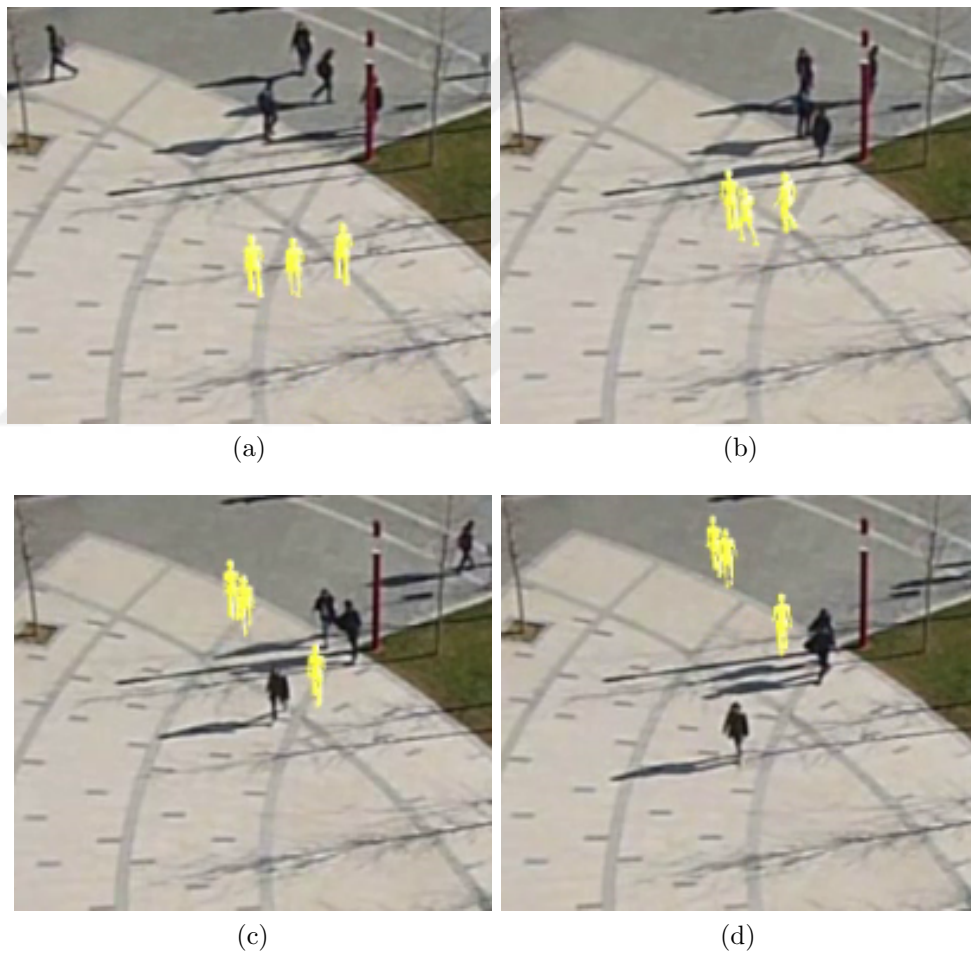


Figure 8.6: The agents in Video-1 drags an artificial agent in (b), which causes it to be left behind in (c) and (d).



Figure 8.7: Sample screen shots from Video-2. The projected, dummy agents are not rendered in the bottom image. The user-controlled agents are shown in yellow.

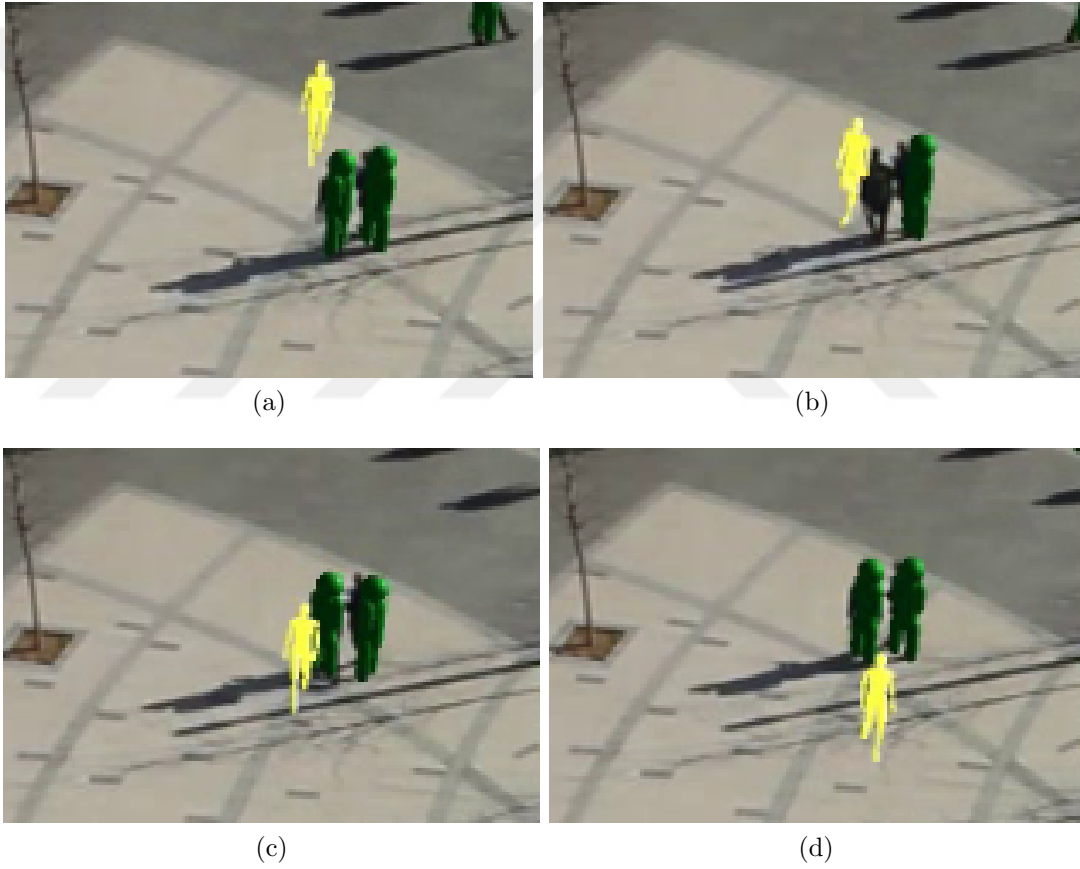


Figure 8.8: A selected artificial agent avoids a couple of projected agents in Video-2. After the projected agent from (b) is recovered in (c), artificial agent continues adjusting its velocity accordingly.



(a)



(b)



(c)



(d)

Figure 8.9: The projected agents enforce the groups of artificial agents to form a line for collision avoidance (Video-2).

Bibliography

- [1] Rhythm and Hues, “Rhythm and Hues: Behind the Scenes of Games Of Thrones Season 5, Ep9 VFX - The Great Fighting Pit,” Accessed 28 Nov. 2018. Available at <https://www.youtube.com/watch?v=WubliIGz2Ls>.
- [2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “SSD: Single shot multibox detector,” in *European Conference on Computer Vision*, ECCV ’16, pp. 21–37, Springer, 2016.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR ’15, pp. 1–9, 2015.
- [5] J. E. Almeida, R. J. Rosseti, and A. L. Coelho, “Crowd simulation modeling applied to emergency and evacuation simulations using multi-agent systems,” *arXiv preprint arXiv:1303.4692*, 2013.
- [6] C. R. Wren, Y. A. Ivanov, D. Leigh, and J. Westhues, “The MERL motion detector dataset,” in *Proceedings of the Workshop on Massive Datasets*, pp. 10–14, ACM, 2007.
- [7] N. Pelechano, J. M. Allbeck, and N. I. Badler, “Virtual Crowds: Methods, Simulation, and Control,” *Synthesis Lectures on Computer Graphics and Animation*, vol. 3, no. 1, pp. 1–176, 2008.

- [8] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 of *CVPR '05*, pp. 886–893, IEEE, 2005.
- [9] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, “Online Multiperson Tracking-by-detection from a Single, Uncalibrated Camera,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 9, pp. 1820–1833, 2011.
- [10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.
- [11] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, *CVPR '14*, pp. 580–587, 2014.
- [12] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, “Selective search for object recognition,” *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.
- [13] R. Girshick, “Fast R-CNN,” in *Proceedings of the IEEE International Conference on Computer Vision*, *ICCV '15*, pp. 1440–1448, 2015.
- [14] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems*, pp. 91–99, 2015.
- [15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, *CVPR '16*, pp. 779–788, 2016.
- [16] S. J. Guy, S. Kim, M. C. Lin, and D. Manocha, “Simulating heterogeneous crowd behaviors using personality trait theory,” in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, *SCA '11*, (New York, NY, USA), pp. 43–52, ACM, 2011.

- [26] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, vol. 96 of *KDD’96*, pp. 226–231, 1996.
- [27] R. J. Campello, D. Moulavi, and J. Sander, “Density-based clustering based on hierarchical density estimates,” in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 160–172, Springer, 2013.
- [28] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.
- [29] A. Rahimi, H. Moradi, and R. A. Zoroofi, “Single image ground plane estimation,” in *Proceedings of the IEEE International Conference on Image Processing*, ICIP ’13, pp. 2149–2153, IEEE, 2013.
- [30] B. Li, K. Peng, X. Ying, and H. Zha, “Vanishing point detection using cascaded 1D Hough Transform from single images,” *Pattern Recognition Letters*, vol. 33, no. 1, pp. 1–8, 2012.
- [31] M. Zhai, S. Workman, and N. Jacobs, “Detecting vanishing points using global image context in a non-Manhattan world,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR ’16, pp. 5657–5665, 2016.
- [32] T. Trocoli and L. Oliveira, “Using the scene to calibrate the camera,” in *Proceedings of the 29th SIBGRAPI Conference on Graphics, Patterns and Images*, SIBGRAPI’16, pp. 455–461, IEEE, 2016.
- [33] F. Lv, T. Zhao, and R. Nevatia, “Camera calibration from video of a walking human,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 9, pp. 1513–1518, 2006.
- [34] J. Liu, R. T. Collins, and Y. Liu, “Surveillance camera autocalibration based on pedestrian height distributions,” in *British Machine Vision Conference*, vol. 2 of *BMVC ’11*, 2011.

- [35] G. M. Brouwers, M. H. Zwemer, R. G. Wijnhoven, and P. H. N. de With, “Automatic calibration of stationary surveillance cameras in the wild,” in *European Conference on Computer Vision, ECCV ’16*, pp. 743–759, Springer, 2016.
- [36] J. Jung, I. Yoon, S. Lee, and J. Paik, “Object detection and tracking-based camera calibration for normalized human height estimation,” *Journal of Sensors*, vol. 2016, 2016. Article no. 8347841, 9 pages.
- [37] D. Liebowitz and A. Zisserman, “Metric rectification for perspective images of planes,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR ’98*, p. 482, IEEE, 1998.
- [38] D. Liebowitz, A. Criminisi, and A. Zisserman, “Creating architectural models from images,” *Computer Graphics Forum*, vol. 18, no. 3, pp. 39–50, 1999.
- [39] K. Chaudhury, S. DiVerdi, and S. Ioffe, “Auto-rectification of user photos,” in *IEEE International Conference on Image Processing, (ICIP ’14)*, pp. 3479–3483, IEEE, 2014.
- [40] B. Bose and E. Grimson, “Ground plane rectification by tracking moving objects,” in *Proceedings of the Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pp. 94–101, 2003.
- [41] A. Bulbul and R. Dahyot, “Populating virtual cities using social media,” *Computer Animation and Virtual Worlds*, vol. 28, no. 5, Article no. e1742, 10 pages, 2017.
- [42] OpenStreetMap, “OpenStreetMap,” Accessed 28 Nov. 2018. Available at <https://www.openstreetmap.org>.
- [43] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2 of *ICCV ’99*, pp. 1150–1157, IEEE, 1999.

- [44] S. Iizuka, Y. Kanamori, J. Mitani, and Y. Fukui, “Efficiently modeling 3d scenes from a single image,” *IEEE Computer Graphics and Applications*, vol. 32, no. 6, pp. 18–25, 2012.
- [45] G. Zhang, X. Qin, X. An, W. Chen, and H. Bao, “As-consistent-as-possible compositing of virtual objects and video sequences,” *Computer Animation and Virtual Worlds*, vol. 17, no. 3-4, pp. 305–314, 2006.
- [46] D. Hoiem, A. A. Efros, and M. Hebert, “Automatic photo pop-up,” *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 577–584, 2005.
- [47] A. Saxena, M. Sun, and A. Y. Ng, “Make3d: Learning 3d scene structure from a single still image,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 824–840, 2009.
- [48] N. M. Thalmann and D. Thalmann, “Animating virtual actors in real environments,” *Multimedia Systems*, vol. 5, no. 2, pp. 113–125, 1997.
- [49] C. Fernández, P. Baiget, F. X. Roca, and J. González, “Augmenting video surveillance footage with virtual agents for incremental event evaluation,” *Pattern Recognition Letters*, vol. 32, no. 6, pp. 878–889, 2011.
- [50] T. Narahara and Y. Kobayashi, “Crowd mapper: Projection-based interactive pedestrian agents for collective design in architecture,” in *Proceedings of the eCAADe 33rd Annual Conference*, pp. 191–200, Education and Research in Computer Aided Architectural Design in Europe (eCAADe), 2015.
- [51] F. Zheng and H. Li, “ARCrowd-a tangible interface for interactive crowd simulation,” in *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI ’11, pp. 427–430, ACM, 2011.
- [52] A.-H. Olivier, J. Bruneau, R. Kulpa, and J. Pettré, “Walking with virtual people: Evaluation of locomotion interfaces in dynamic environments,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 7, pp. 2251–2263, 2018.
- [53] P. Baiget, C. Fernández, X. Roca, and J. González, “Generation of augmented video sequences combining behavioral animation and multi-object

- tracking,” *Computer Animation and Virtual Worlds*, vol. 20, no. 4, pp. 473–489, 2009.
- [54] J. I. Rivalcoba, O. De Gyves, I. Rudomin, and N. Pelechano Gómez, “Coupling camera-tracked humans with a simulated virtual crowd,” in *Proceedings of the 9th International Conference on Computer Graphics Theory and Applications*, GRAPP ’14, pp. 312–321, SciTePress, 2014.
- [55] Unity Team, “Unity,” Accessed 28 Nov. 2018. Available at <http://unity3d.com/>.
- [56] OpenCV Team, “OpenCV (Open Source Computer Vision Library),” Accessed 28 Nov. 2018. Available at <http://opencv.org>.
- [57] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [58] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, “Mot16: A benchmark for multi-object tracking,” *arXiv preprint arXiv:1603.00831*, 2016.
- [59] Z. Zivkovic and F. van der Heijden, “Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction,” *Pattern Recognition Letters*, vol. 27, no. 7, pp. 773–780, 2006.
- [60] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, vol. 2 of *IJCAI ’81*, (Vancouver, BC, Canada), pp. 674–679, 1981.
- [61] J. Shi and C. Tomasi, “Good Features to Track,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR’94, pp. 593–600, IEEE, 1994.
- [62] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

- [63] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR ’17, pp. 3296–3305, 2017.
- [64] D. Kurtaev, “OpenCV TensorFlow Object Detection API,” Accessed 28 Nov. 2018. Available at <https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API>.
- [65] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR ’16, pp. 2818–2826, 2016.
- [66] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.
- [67] G. Welch and G. Bishop, “An Introduction to the Kalman Filter,” tech. rep., University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995.
- [68] Wikipedia, “Image moment,” Accessed 28 Nov. 2018. Available at https://en.wikipedia.org/wiki/Image_moment.
- [69] J. Ferryman and A. Ellis, “PETS2010: Dataset and challenge,” in *Proceedings of the 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, AVSS ’10, pp. 143–150, Aug 2010.
- [70] L. Leal-Taixé, A. Milan, I. D. Reid, S. Roth, and K. Schindler, “MOTChallenge 2015: Towards a benchmark for multi-target tracking,” *CoRR*, vol. abs/1504.01942, 2015. Available at <http://arxiv.org/abs/1504.01942>.
- [71] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *European Conference on Computer Vision*, ECCV ’14, pp. 740–755, Springer, 2014.

- [72] R. Stiefelhagen, K. Bernardin, R. Bowers, J. Garofolo, D. Mostefa, and P. Soundararajan, "The CLEAR 2006 evaluation," in *International Evaluation Workshop on Classification of Events, Activities and Relationships*, pp. 1–44, Springer, 2006.
- [73] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the CLEAR MOT metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, p. 1, 2008.
- [74] Scikit-learn.org, "Bandwidth estimator (sklearn.cluster.estimate_bandwidth)," Accessed 28 Nov. 2018. Available at http://scikit-learn.org/stable/modules/generated/sklearn.cluster.estimate_bandwidth.html.
- [75] S. Murali and V. Govindan, "Shadow detection and removal from a single image using lab color space," *Cybernetics and information technologies*, vol. 13, no. 1, pp. 95–103, 2013.
- [76] S. Murali and V. Govindan, "Removal of shadows from a single image," in *Proceedings of First International Conference on Futuristic Trends in Computer Science and Engineering*, vol. 4, pp. 111–114, 2006.
- [77] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [78] N. P. Walsh, "The tallest residential building in the world is coming to New York city," *ArchDaily - Broadcasting Architecture Worldwide*, Accessed 28 Nov. 2018. Available at <https://www.archdaily.com/904133/the-tallest-residential-building-in-the-world>.
- [79] J. Semple and G. Kneebone, *Algebraic Projective Geometry*. Oxford University Press, Oxford, 1979.
- [80] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Numerical Analysis*, pp. 105–116, Springer, 1978.

- [81] J. Shewchuk, “Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator,” *Applied Computational Geometry Towards Geometric Engineering*, pp. 203–222, 1996.
- [82] K. Wurster, H. Butler, O. Tonnhofer, S. Gillies, and J. Arnott, *Shapely - Manipulation and Analysis of Geometric Objects*. The Toblerity Project, Accessed 28 Nov. 2018. Available at <https://github.com/Toblerity/Shapely>.
- [83] Google Maps, “University of Reading - Google Maps,” Accessed 28 Nov. 2018. Available at <https://www.google.co.uk/maps/place/University+of+Reading/@51.4385773,-0.9446929,116m/data=!3m1!1e3!4m5!3m4!1s0x487684b360159d63:0xa74d6f19cd5e05ca!8m2!3d51.4414205!4d-0.9418157?hl=en>.
- [84] Yandex, “Map of Bilkent University, Computer Engineering Department,” Accessed 28 Nov. 2018. Available at <https://yandex.com.tr/harita/11503/ankara/?l=sat&ll=32.750607%2C39.870842&mode=search&sctx=ZAAAAAgBEAAaKAoSca8mT11NbUBAEcmTpGum9kNAEhIJUQMGAIAg1D8RXItF%2FX0gvz8iBAABAgQoCjAA0Jfb1pjG14CqgwFA711IAVXNzMw%2BWABqAnRycACdAc3MzD2gAQCoAQA%3D&sll=32.750607%2C39.870842&sspn=0.002457%2C0.000951&text=bilkent&z=19>.
- [85] X. Cui and H. Shi, “A*-based pathfinding in modern computer games,” *International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 125–130, 2011.
- [86] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA ’08, pp. 1928–1935, IEEE, 2008.
- [87] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics research*, pp. 3–19, Springer, 2011.
- [88] ROCKVR Team, “Vr capture,” 2018. Available at <https://assetstore.unity.com/packages/tools/video/vr-capture-75654>.

- [89] S. Narang, T. Randhavane, A. Best, A. Shapiro, and D. Manocha, “FBCrowd: Interactive multi-agent simulation with coupled collision avoidance and human motion synthesis,” tech. rep., Department of Computer Science, University of North Carolina at Chapel Hill, 2016.



Appendix A

Pedestrian Tracking Results

Tracking results including detection boxes and pedestrian postures for the test videos can be seen in Figures A.1, A.2, and A.3. HOG, *Inception* and *MobileNet* are ordered from top to bottom.

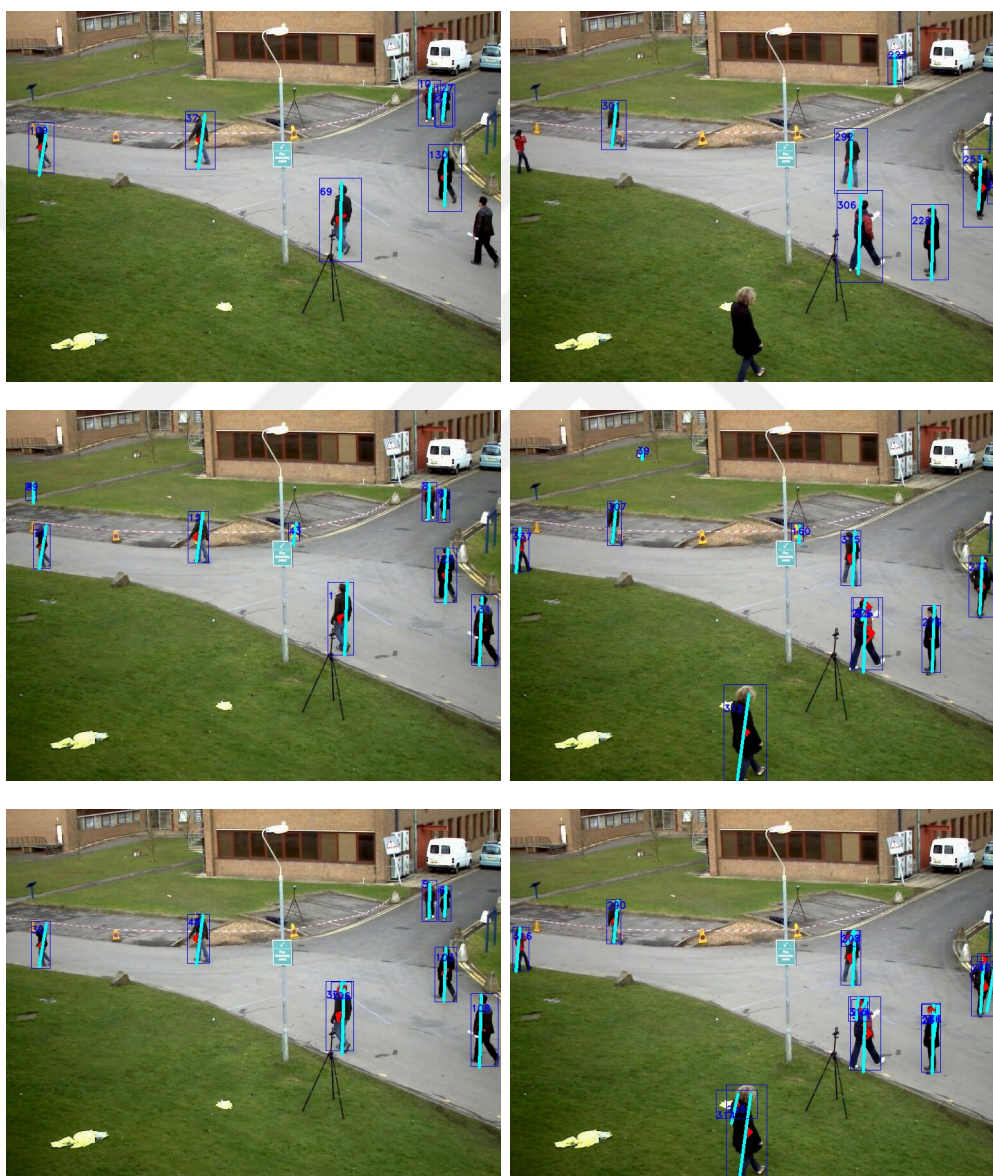


Figure A.1: Further example pedestrians postures from PETS video.

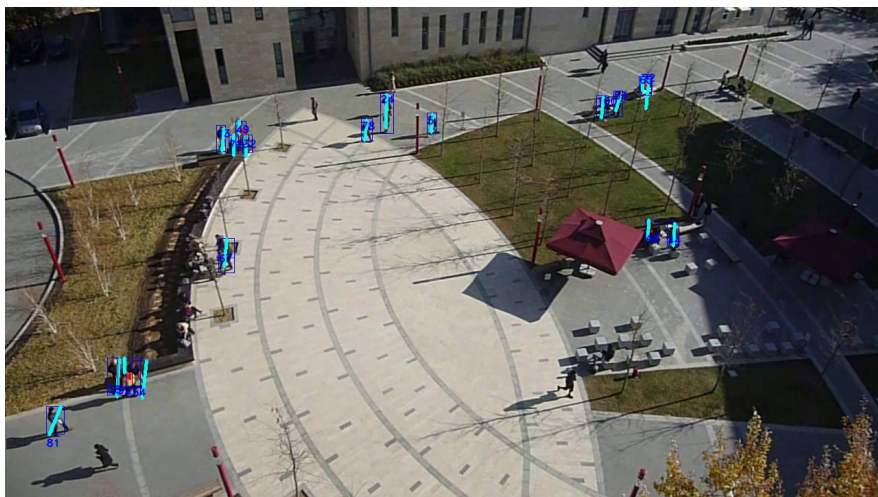
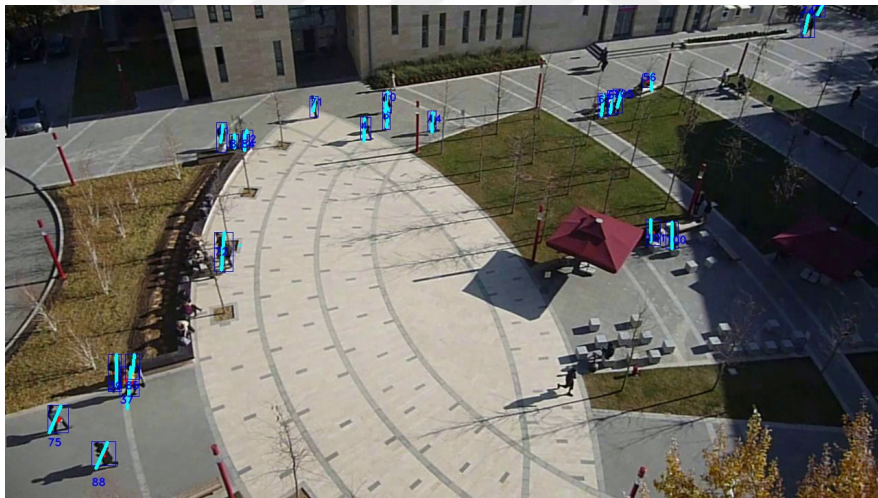
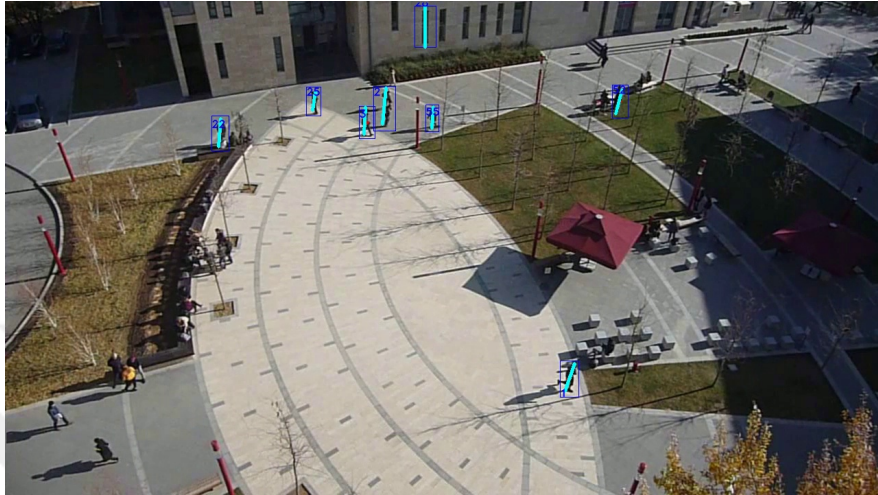


Figure A.2: Example pedestrians postures from Video-1.

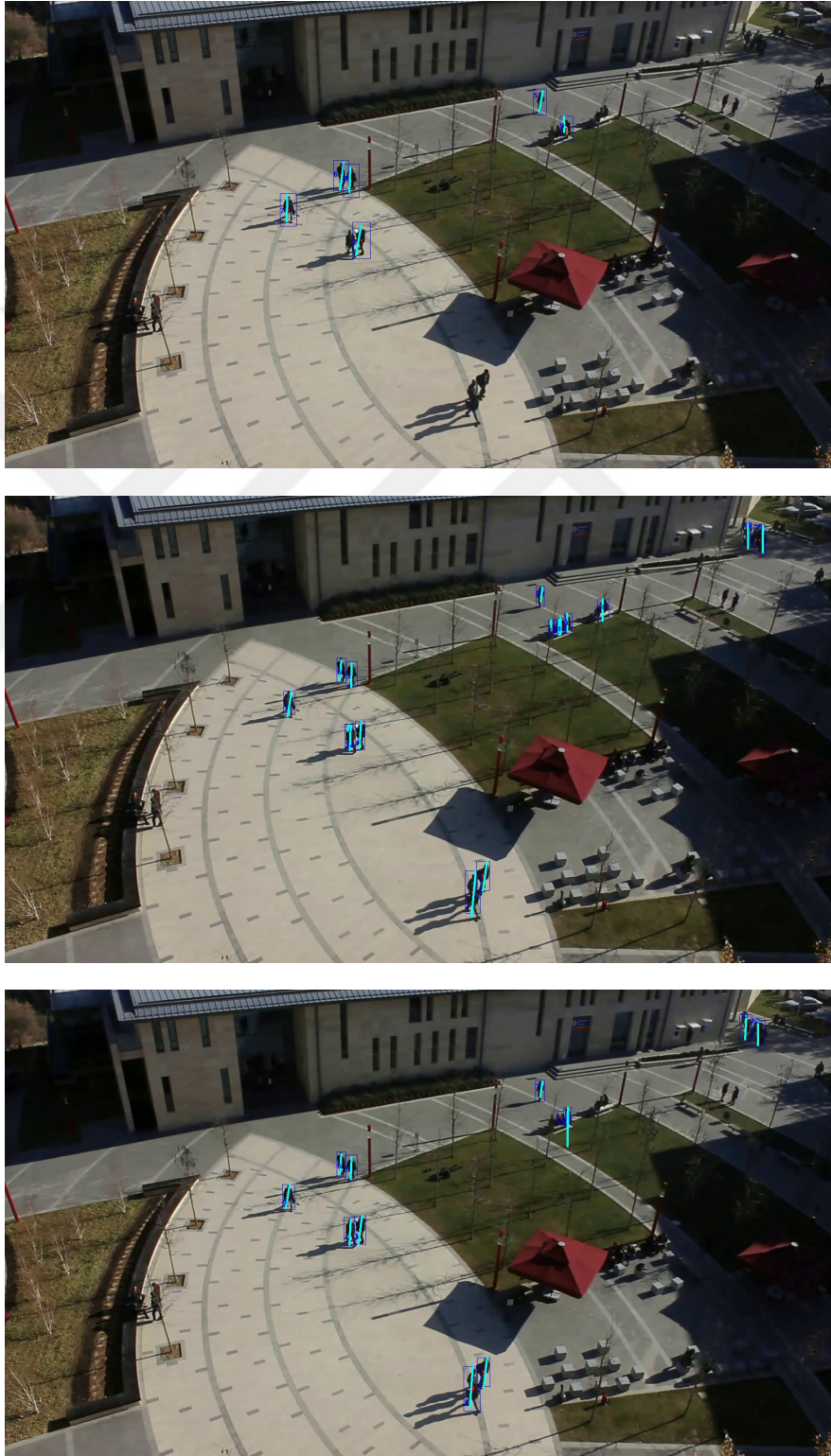


Figure A.3: Example pedestrian postures from Video-2.