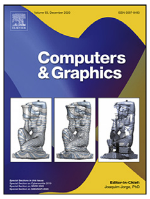
Computers & Graphics 95 (2021) 141–155

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag



Technical Section

An augmented crowd simulation system using automatic determination of navigable areas[☆]

Yalım Doğan[a], Sinan Sonlu[a], Uğur Güdükbay[a,*]

Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey

ARTICLE INFO

Article history:
Received 29 June 2020
Revised 20 January 2021
Accepted 28 January 2021
Available online 10 February 2021

MSC:
65D18
65D19

Keywords:
Pedestrian detection and tracking
Data-driven simulation
Three-dimensional reconstruction
Crowd simulation
Augmented reality
Deep learning

ABSTRACT

Crowd simulations imitate the group dynamics of individuals in different environments. Applications in entertainment, security, and education require augmenting simulated crowds into videos of real people. In such cases, virtual agents should realistically interact with the environment and the people in the video. One component of this augmentation task is determining the navigable regions in the video. In this work, we utilize semantic segmentation and pedestrian detection to automatically locate and reconstruct the navigable regions of surveillance-like videos. We place the resulting flat mesh into our 3D crowd simulation environment to integrate virtual agents that navigate inside the video avoiding collision with real pedestrians and other virtual agents. We report the performance of our open-source system using real-life surveillance videos, based on the accuracy of the automatically determined navigable regions and camera configuration. We show that our system generates accurate navigable regions for realistic augmented crowd simulations.

1. Introduction

Crowd simulations investigate the interaction of individuals inside and among groups of people, in terms of behavior, appearance, personality, and emotions. Models used in such simulations aim for a realistic interaction with the environment; hence, the appearance and behavior of the virtual agents that represent individuals should fit the context of the scene for better immersion. Quantitative methods assess the realism of such simulations, comparing the simulated crowd with real-world data.

Augmenting virtual crowds into real-life videos has applications in entertainment, security, and education. Virtual crowds can cost-effectively fill environments in movies, appearing together with real actors; virtual tutors can move inside live environments to create immersion in training applications. In such augmented crowd simulations, virtual agents should be indistinguishable from the real people and should interact with the real crowd and the environment realistically. This requires careful inspection of the environment and the individuals in the video.

Augmented crowd simulations benefit from data-driven approaches for pedestrian and scene inference. Using a model for the environment and pedestrian trajectories, a virtual crowd can be augmented into the input video, so that virtual agents plausibly move in the scene without colliding with each other and the real pedestrians. However, in such a workflow, many steps require labor-intensive manual processing, including the construction of an environment model for the virtual crowd.

We introduce our open-source augmented crowd simulation system that utilizes an automated approach for the determination and reconstruction of navigable regions in real-life surveillance-like videos. We make use of existing methods of semantic segmentation and pedestrian tracking to determine image-level navigable regions. Then we reconstruct the aerial view of these regions as a flat mesh and position it in our 3D crowd simulation environment. From the perspective of the automatically calibrated scene camera, the virtual agents move inside the navigable regions of the video, avoiding scene obstacles, real pedestrians, and other virtual agents. We evaluate the accuracy of the generated navigable regions in comparison to the ground truth, using real-life surveillance videos.

We list our contributions as:

- Automatic determination and reconstruction of image-level navigable areas in surveillance-like videos for seamless integration of virtual agents.

[☆] This paper was recommended for publication by Stefanie Zollmann.

[*] Corresponding author.

E-mail addresses: yalim.dogan@bilkent.edu.tr (Y. Doğan), sinan.sonlu@bilkent.edu.tr (S. Sonlu), gudukbay@cs.bilkent.edu.tr (U. Güdükbay).

https://doi.org/10.1016/j.cag.2021.01.012
0097-8493/© 2021 Elsevier Ltd. All rights reserved.

- Evaluation of the resulting image-level navigable areas using different combinations of segmentation networks and training sets.

2. Related work

2.1. Data-driven crowd simulations

Many applications of crowd simulations utilize real-life data for realistic agent behavior. Musse et al. [1], Lerner et al. [2], and Kim et al. [3] extract pedestrian trajectories from real-life video sequences to simulate movements of virtual agents in various crowd scenarios. Jablonski et al. [4] evaluate the accuracy and the realism of crowd simulations in comparison to real-life footage, using pedestrian flow. Amirian et al. [5,6] generate crowd trajectories that mimic the behavior of real-life pedestrians, regarding their interaction with the environment and other pedestrians. Bera et al. [7] learn pedestrian motion models from video to simulate agents that act like real pedestrians. Instead of learning the behavior of the real crowd, we utilize the pedestrian data to automatically determine the navigable regions of the scene to realistically integrate virtual agents.

2.2. Augmenting virtual agents into real-life videos

Numerous works augment virtual agents into videos of real people and environments. Zheng and Li [8] manipulate virtual crowds using an augmented reality interface. Baiget et al. [9] generate augmented video sequences with virtual crowds that react to the environment and people, utilizing multi-object tracking to detect dynamic entities in the video feed. They generate the environment using a calibrated static camera and model the context-aware behaviors of the agents using Situation Graph Trees. In contrast to our approach, they do not generate the navigable area based on the input video, and they do not use collision avoidance for the interaction of virtual agents with real pedestrians and other obstacles in the video. Instead, they generate paths on the ground for specific behaviors such as walking in a circle or a spiral path. Olivier et al. [10] 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.

Rivas et al. [11] propose a framework for coupling simulated crowds with real pedestrians on completely navigable environments. Zhang et al. [12] propose a framework for the seamless integration of virtual agents into videos where agents avoid collision with real pedestrians. They use an alpha map-based solution to handle occlusion between agents and real pedestrians. Doğan et al. [13] track pedestrians using a HOG-based detector and Kalman filtering, to augment virtual agents on manually constructed navigable areas. In contrast, we analyze the video to extract the navigable regions automatically.

2.3. Reconstructing navigable regions

The reconstruction of navigable regions requires a geometric understanding of the scene. To this end, an image-based horizon approximation can give an idea about the camera configuration. To find the horizon and a suitable third vanishing point, Li et al. [14] use intersections of Hough lines, whereas Zhai et al. [15] use deep convolutional neural networks. Trocoli and Oliveira [16] generate a histogram of angles using the line segments of the image and use the peaks to search for vanishing points. These methods rely on the existence of horizontal lines, which usually exist in man-made structures, but not guaranteed in every input scene.

In contrast, various studies utilize pedestrian features to calculate vanishing points, treating pedestrians as vertical poles intersecting at the third vanishing point. Pedestrian trajectories are then used to calculate the remaining vanishing points on the horizon. Liu et al. [17] use pedestrian postures to find the third vanishing point and use this information for automatic camera calibration. Similarly, Brouwers et al. [18] consider the height distribution of the pedestrians to calculate the tilt angle of the camera. Jung et al. [19] estimate normalized pedestrian height using tracking based camera calibration. We utilize both line and pedestrian features for camera calibration, which performs better in arbitrary scenes.

After calibration, we generate the blueprint (the top-down view from an imaginary camera that is orthogonal to the ground plane) of the navigable regions for reconstruction, which requires metric rectification. Liebowitz et al. [20,21] apply metric rectification to correct distorted perspective in images, and reconstruct architectural scenes in 3D. Bose and Grimson [22] rectify the ground plane by tracking the moving objects in the scene. Chaudhury et al. [23] use line properties to find two vanishing points in general images, to perform affine rectification that corrects the perspective distortion partially, without recovering the real angles. In our blueprint generation, we utilize the approaches of Bose and Grimson [22], and Chaudhury et al. [23].

Various works that reconstruct the scene in 3D using a single image or a static camera are also worth mentioning. Bulbul and Dahyot [24] use social media location data to populate manually built 3D cities. Iizuka et al. [25] use manual annotation of specific boundaries in a single image to reconstruct the 3D scene. Zhang et al. [26] automatically reconstruct the 3D scene based on the epipolar geometry of multiple views. Hoiem et al. [27] and Saxena et al. [28] use single view approaches that reconstruct the 3D scene automatically. Although we place virtual agents in a 3D environment, the reconstructed navigation mesh is flat, which is sufficient for projecting virtual agents onto the video.

3. Framework

Our open-source framework, outlined in Fig. 1, provides an augmented interactive crowd simulation in Unity [29]. We simulate virtual agents walking in navigable regions of the input video while avoiding collision with real pedestrians. To reconstruct the navigable scene, we preprocess the input video using computer vision techniques included in the OpenCV library [30]. The crowd simulation runs in real-time, and the preprocessing is performed off-line.

The input of our system is the video of an environment containing pedestrians. We assume this video is recorded with a stationary camera, using an angle similar to surveillance footage. The number of active pedestrians in the scene increases the accuracy of the reconstruction. The main stages of our workflow are as follows:

1. *Pedestrian Detection and Tracking:* We detect the pedestrians in the video and track their positions in consecutive frames. We record the tracking data, including pedestrian locations and postures, into a MOT [31] compatible file.
2. *Navigable Region Calculation:* We first apply semantic segmentation on multiple frames of the input video to form navigable region candidates. We filter out the obvious non-navigable regions such as walls. We then use pedestrian information to evaluate each region. We accept regions with sufficient pedestrian occupancy as navigable.
3. *Scene Reconstruction:* We extract the vanishing points from the video and apply perspective correction on the navigable areas to obtain an aerial view of the navigable scene. We use the

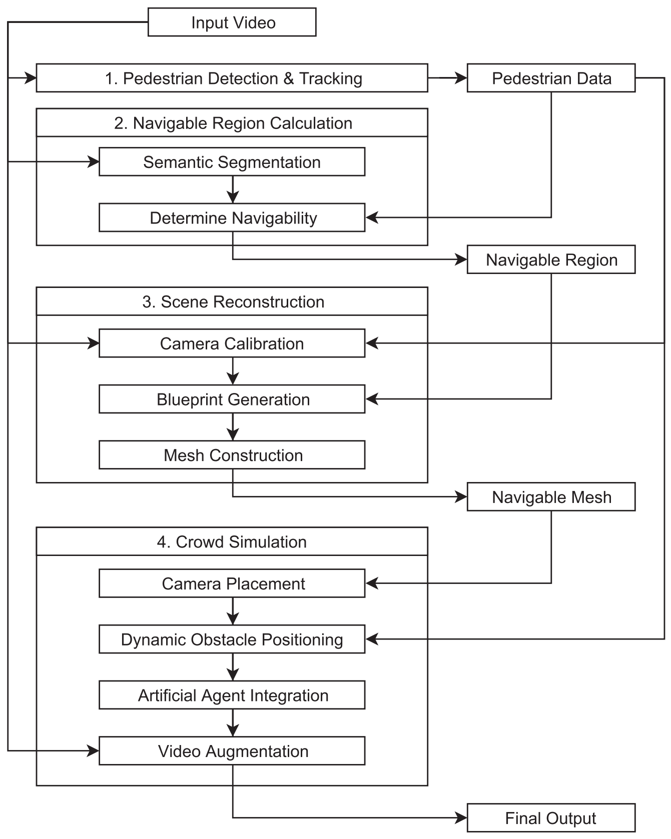


Fig. 1. The augmented crowd simulation framework.

term “blueprint” for this top-down view. We utilize an iterative Perspective-n-Point (PnP) solution to correct the imperfect blueprint to match its real-life counterpart. We then convert the blueprint to a 2D mesh.

4. *Crowd Simulation:* We place the 2D navigation mesh into Unity and position the 3D scene camera so that the navigable regions of the video overlaps with the projection of the 2D mesh. We project real pedestrians in the video that are detected in Pedestrian Detection and Tracking stage onto the navigable mesh as dynamic obstacles. The user can insert virtual agents into the scene and set their destination on the navigable region with mouse input. Each agent walks towards its destination while avoiding collision with the projected pedestrians and other agents.

We discuss our approach for reconstructing the navigable regions in Section 4. The following subsections describe the other stages in more detail.

3.1. Pedestrian detection and tracking

We first preprocess the input video to extract the pedestrian information. Although we assume the video is recorded with a stationary camera, there may be minor position or orientation changes caused by environmental factors. To eliminate such disruptions, we stabilize the video using optical flow [32].

We process the video using one of the available pedestrian detection techniques including Histogram of Oriented Gradients (HOG) [33] and one-shot SSD networks [34]: either Inception v2 [35,36] and MobileNet [37]. We use pretrained versions of these networks available in OpenCV Tensorflow API [38,39].

To reduce processing time, we assume anything immobile is not a pedestrian. This also eliminates idle pedestrians; we find this acceptable because their role in collision avoidance is minimal. We detect movement using background subtraction [40] and look for pedestrians only in these areas.

We detect the pedestrians in the current frame and update the tracking information using the pedestrians detected in preceding frames. We use Kalman Filtering [41] to minimize position jumps caused by noisy detection. At the end of this stage, we only report consistently tracked pedestrians. Tracking information contains pedestrian positions per frame, including the head and foot positions of each pedestrian. We define head and foot positions as the endpoints of the foreground pixel blob which encapsulates the detected pedestrian. We use this pedestrian data in the navigable region calculation, camera calibration, and dynamic obstacle positioning. We use foot positions to project dynamic obstacles in place of real pedestrians. We use the combination of head and foot positions as vertical poles for camera calibration and to determine average pedestrian height in simulation.

3.2. Augmented crowd simulation

We simulate virtual agents in the reconstructed scene that avoid real pedestrians. We generate a variety of realistic human models using MakeHuman [64] to act as virtual agents. We utilize pathfinding techniques included in Unity to generate a navigation mesh from the reconstructed scene. For collision avoidance, we use Reciprocal Velocity Obstacles (RVO) [42,43]. RVO is not included in Unity's pathfinding tools; therefore, we use the RVO implementation available at [65]. We use Unity's pathfinding tools for global path planning over the calculated navigation mesh where each virtual agent follows a trajectory with sub-goals to the desired position. We utilize RVO for local path planning, i.e., collision detection and avoidance, where we dynamically adjust the velocities of virtual agents towards the next goal position to avoid colliding with other agents. In traditional RVO, each agent is responsible for adjusting its velocity and direction to prevent oscillations. In augmented crowd simulations, the real pedestrians follow predetermined paths; only virtual agents actively participate in collision avoidance. We integrate real pedestrians into the simulation by setting the maximum number of considered neighboring agents in RVO to zero. As a result, the real pedestrians simply act as dynamic obstacles without any change to their velocity during collision avoidance.

We project the pedestrians onto the navigable mesh by sending camera rays through their feet positions on the image plane. We refresh the projection in every frame and update the position of these dynamic obstacles formed by the real pedestrians. If a projection becomes out of sync with the associated pedestrian's tracking information, it means the tracker lost the pedestrian, therefore we remove the corresponding dynamic obstacle after a few frames. The projected dynamic obstacle uses its latest displacement as its current movement vector.

We determine the height of a virtual agent based on the average height of the pedestrian detection boxes, projected to the 3D scene. We send a camera ray r to the head position of the pedestrian in the image plane. Given the 3D location of the camera C and the feet position F, we determine the head location H using Eq. (1), where D is the vector between the camera and the feet location on the XZ-plane. We calculate the distance from the camera to the head in terms of the ray unit vector $\hat{r}$ using D and add it to the camera location. The height is relative to the unit distance in Unity. We calculate the height at the beginning of the simulation for an initial approximation, but it can be specified by the user at run time, considering the existing bounding boxes.

$$\begin{aligned} D &= (F_x, 0, F_z) - (C_x, 0, C_z), \\ H &= \hat{r}\,\frac{\|D\|}{\hat{r}\cdot D} + C. \end{aligned} \tag{1}$$

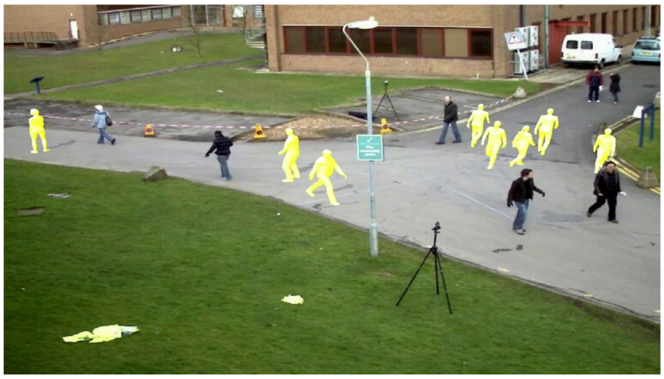


Fig. 2. The screenshot shows a reconstructed scene including real pedestrians and virtual agents (in yellow) walking around without colliding with pedestrians or each other. The original scene is courtesy of PETS09-S2L1 video [44]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Our augmented crowd simulation framework starts by inputting the stabilized video, the pedestrian tracking and camera calibration data, and the reconstructed navigable mesh. After placing the camera and the navigable mesh, the simulation starts by projecting the video as a background in the camera frustum, and the virtual agents are visible on top of the navigable regions. Fig. 2 shows a screenshot of a reconstructed scene including real pedestrians and virtual agents. In the following section, we focus on the automatic determination of navigable areas for augmented crowd simulations.

4. Navigable area reconstruction

We infer the navigable regions of the scene and generate a 2D navigation mesh based on the union of these regions in an aerial view. We position the 2D mesh into our simulation environment with the camera configuration of the video, so that the virtual agents that walk on the navigation mesh appear as if they are walking on the navigable regions of the video. This reconstruction process involves the following steps:

1. We analyze the video frames to determine the navigable regions using deep-learning-based semantic segmentation and pedestrian tracking.
2. We determine the scene geometry by detecting the horizon and the third vanishing point, using line features from the first frame and pedestrian data. We then use homography to generate the "blueprint" of the navigable regions.
3. We orient the camera in the reconstructed scene according to the perspectively-corrected navigable regions. Because the data used in the previous step contains noise, the perspective correction is not perfect, which also affects camera placement. For this reason, perspective correction is readjusted to match the real-life counterpart for optimum camera placement.
4. We then convert the "blueprint" of the final corrected navigable regions from an image to a 2D mesh that can be placed in the 3D scene in Unity.

In the following subsections, we discuss each step in detail.

4.1. Semantic segmentation for navigable region detection

The first step of the reconstruction process is to determine the navigable regions using multiple video frames. Even though the video is stationary, using only the first frame is not enough, since walking pedestrians would obscure navigable areas behind.

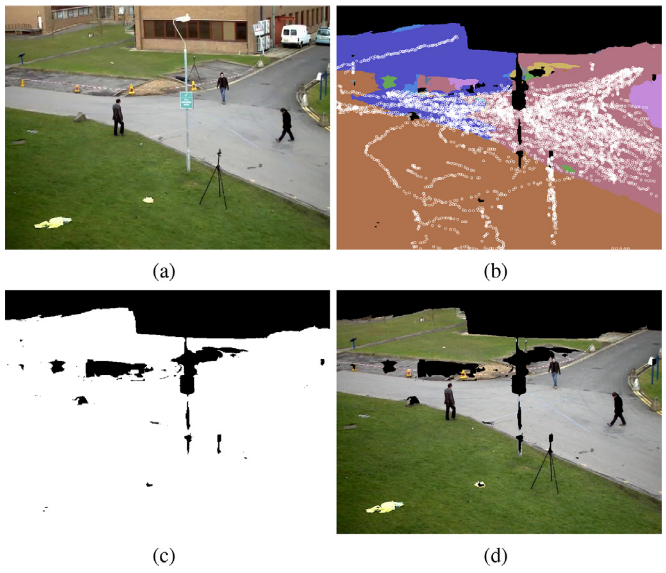


Fig. 3. Navigable region calculation of PETS09-S2L1 [44] using Xception network for segmentation. (a) the original frame, (b) the navigable region candidates with pedestrian trajectories (in white), (c) union of the navigable regions (in white), (d) navigable regions visualized on the original frame.

Therefore we update the navigable region map every N frames (user-defined) by accumulating the segmented regions. To improve segmentation performance for videos with hard shadows where context is hardly distinguishable, we apply the shadow removal method by Murali and Govindan [45,46].

We segment the frames and update navigable region candidates using one of the state-of-the-art semantic segmentation methods: MobileNetV2 [47] or Xception [48,49]. We associate each segment with a label, which helps us filter out the obvious non-navigable areas such as walls, buses, and cars. The labels are different for each dataset that we use in training. We used networks pre-trained [50,51] on ADE20K [52,53] and Cityspaces [54] datasets. In the evaluation section, we experiment on different input videos using various combinations of segmentation networks and training sets.

We determine the "navigability" of a region not only by its semantic label but also by considering the pedestrian data. A particular region is navigable if the ratio of frames in which the region is occupied by at least one pedestrian exceeds a user-defined threshold. This threshold will be referred to as the "navigation density threshold" for the rest of the work. We assess this per-region instance, therefore identifying a region of a particular label as navigable does not change the navigability of other disconnected regions with the same label. In the absence of any pedestrians, we determine the navigability based on the semantic label only. Fig. 3 shows the navigable regions extracted from PETS09-S2L1 [44].

4.2. Perspective correction for blueprint generation

To reconstruct a 2D navigation mesh using the image of the navigable regions, we first obtain their "blueprint", i.e., the rectified aerial view. To this end, we utilize various computer vision techniques, including RANSAC-based horizon detection, homography-based perspective correction, and the solutions for the PnP problem.

4.2.1. Horizon detection

We first find the vanishing points in the scene to extract the scene structure observed from the camera viewpoint. In a given image, it is possible to find infinitely many vanishing points; in our case, we are interested in finding only the three: two on the horizon and a third one that is not on the horizon. The position of the third vanishing point concerning the horizon depends on the placement of the camera, specifically its tilt angle.

We assume the camera is looking down on the scene in a surveillance-like configuration, in which it is easier to generate the navigable areas in comparison to pedestrian level camera placements. Besides, the areas below the horizon have more coverage in this configuration, enabling a better view of the ground plane. In this configuration, the horizon is above the image center and the third point is below it. We refer to the third point as "nadir" from this point forward.

To find the vanishing points, we utilize the lines found in the image, similar to Li et al. [14]. The lines that are parallel in the real world, which can be found abundantly in man-made scenes, intersect at a vanishing point in the image. The intersections of horizontal lines are positioned on the horizon and the intersections of vertical lines give us the third vanishing point. We can easily find such lines by filtering and Hough transform. Given four points i, j, u, v in the 2D image, where $\overline{ij}$ is parallel to $\overline{uv}$ in 3D space, we find each vanishing point using homogeneous coordinates following Eq. (2). To find the horizon, we calculate multiple vanishing points using different line pairs, that are all parallel in the 3D space. We then find a line that connects these vanishing points to determine the horizon.

$$\begin{aligned} p_i &= [i_x, i_y, 1], \\ p_j &= [j_x, j_y, 1], \\ l_{ij} &= p_i \times p_j, \\ vp &= l_{ij} \times l_{uv}. \end{aligned} \tag{2}$$

In cases where the visual cues of the image background are not sufficient, we use pedestrian data to calculate vanishing points [17–19,55]. We treat pedestrians as vertical poles that change position between frames, assuming they do not change their stance too much. We use their postures as parallel vertical lines that converge at the nadir vanishing point. We use their location changes to generate trajectories to be used as additional line features. Having the head and foot positions of the pedestrians in each frame, we sample the head and foot trajectories of each pedestrian at user-defined intervals. We assume that the height of each pedestrian is constant throughout the video; hence, we take the aforementioned lines to be parallel in the real world and meet at the horizon in the image (cf. Fig. 4).

Because the pedestrian data and lines from the image may contain noise, we use RANSAC [56] and thresholding to reduce the noise. We only consider a subset of the given lines based on length [23]. For every random line combination, we calculate a score 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 equivalent to its length. We use the model with the highest count as the vanishing point. Additionally, we enforce the horizon vanishing points to be above any line segment from pedestrian trajectories, which ensures that the navigable area stays below the horizon. Similarly, the nadir point is expected to be below the trajectories.

For the camera calibration, we need to determine its intrinsic parameter matrix (K). In K, we assume zero skew ($s = 0$) and take α as the aspect ratio of the image. We calculate the focal length f using the orthocenter of the triangle defined by the vanishing points as $f^2 = |vp_1 - p||vp_2 - p| - |o - p|^2$, where o is the orthocenter, p is the projection of the orthocenter on the horizon, and vp_1 and vp_2 are the vanishing points on the horizon.

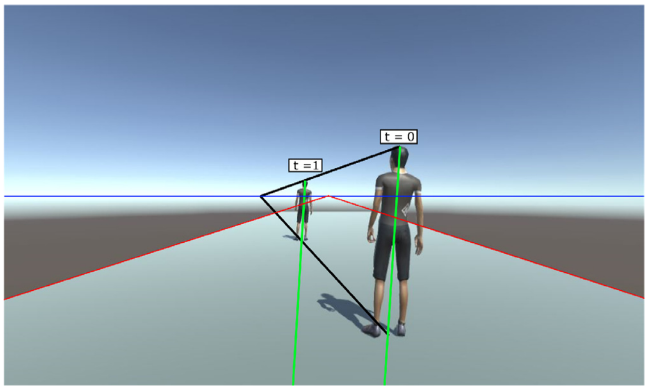


Fig. 4. The head and foot positions of the pedestrian at different frames create parallel lines in the real world (shown in black), which defines a single vanishing point at their intersection. We extract the red lines from the image (which are parallel in the 3D scene) and define another vanishing point where they together define the horizon, shown in blue. Additionally, we use the postures (head to foot vertical line of each pedestrian, shown in green) to find the nadir vanishing point, which cannot be seen in the example because it is too far away. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.2.2. Image rectification

We use the vanishing points for perspectively correcting the segmented image. To this end, we rectify the image by projectively warping it as if the generated image is taken from a frontal, bird's eye view angle. When we take an image from such an angle, the view plane is parallel to the navigation area in the frame. To rectify an image, we use homography [57]. Given points in one plane as x, we calculate the corresponding points in the second plane as $x' = Hx$ where H is the Homography matrix.

To construct H, we need to have at least four-point correspondences between two planes. In our scenarios, we do not have such point correspondences as they require knowledge of the scene, such as a window with a known shape in the real world. Another way of constructing H, called *stratified rectification* [20,21], is to look at its decomposition (see Eq. (3)). The leftmost matrix H_s is the similarity matrix (metric part), which contains rotation (r), translation (t), and scaling (s) components. H_a is the affine transformation matrix, and H_p is the projective transformation matrix.

$$\begin{aligned} H &= H_s H_a H_p \\ &= \begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\beta & -\alpha/\beta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & 1 \end{bmatrix}. \end{aligned} \tag{3}$$

We construct H starting from the projective transformation matrix. In Eq. (3), 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 the horizon. Applying the projection matrix recovers the parallelism of the lines in the image (affine rectification). The next step is to recover the metric properties of the image such as the length ratios and angles of non-parallel lines using the affine transformation matrix H_a. We calculate the parameters α and β in H_a using the concept of circular points [58].

Each method defines circles with center $(c_\alpha, 0)$ and radius r, where the first axis is α and the second is β. The intersections of the circles determine the affine parameters α and β. To find those circular points, Bose and Grimson [22] utilize the trajectories of moving objects to be used for ratios of lengths in the image. They use lines in 4 and 5 to calculate the centers and radius of each circle. They define each line using two points p_1 and p_2, where

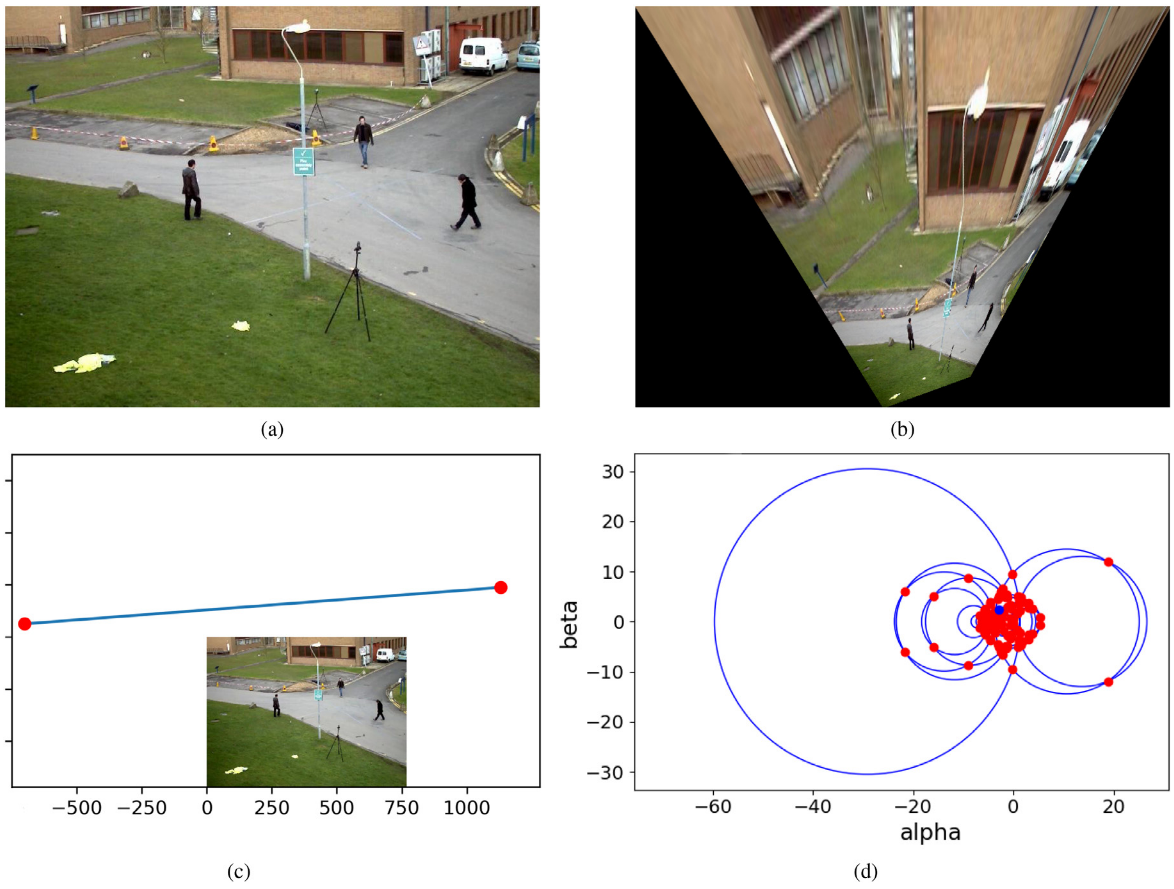


Fig. 5. The overview of the metric rectification process for PETS09-S2L1. The original image (a). After determining the horizon for the image to calculate the projective transformation (c), we use the pedestrian trajectories to find the circular points to be used in affine transformation (d). We apply the resulting homography matrix to warp the image as if the generated image is taken from a bird's eye view (b).

s is the length ratio, $\Delta x = p_{1x} - p_{2x}$ and similarly for Δy. We utilize the feet trajectories of pedestrians as non-parallel paths in the image. We assume pedestrians have the same velocity in the real world and took all paths with a constant velocity. Therefore, s is taken as 1. Because there are many intersection points for circles, we take their average. We then use the resulting point (α, β) in the affine transformation matrix H_a.

$$(c_\alpha, c_\beta) = \left(\frac{\Delta x_1 \Delta y_1 - s^2 \Delta x_2 \Delta y_2}{\Delta y_1^2 - s^2 \Delta y_2^2}, 0 \right), \tag{4}$$

$$r = \left| \frac{s(\Delta x_2 \Delta y_1 - \Delta x_1 \Delta y_2)}{\Delta y_1^2 - s^2 \Delta y_2^2} \right|. \tag{5}$$

We apply similarity transformations to our resulting metric-rectified image, as described by Chaudhury et al. [23] to keep its features within the image boundaries. We use the implementation provided by Chilamkurthy [59]. Fig. 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.

4.3. Camera placement

After applying metric rectification on our segmented image to obtain the aerial view, we determine the orientation of our camera so that the navigable model's projection on the view plane matches its real-world counterpart. We use the pinhole camera model for projection, which projects the 3D points (X, Y, Z) in the world scene to 2D image points (u, v) based on the projection matrix P in Eq. (7). K is the intrinsic matrix of the camera, and the extrinsic part of the camera model is $[R|t]$. We describe how we approximate the extrinsic part of the camera model, as we have already constructed the intrinsic matrix using the vanishing points.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = P \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{6}$$

$$P = K[R|t], \tag{7}$$

To find the transformation expressed in the extrinsic matrix, we utilize PnP solutions that use point correspondences between the 2D image points and the 3D world points to approximate camera orientation. A stable implementation of an iterative PnP solver is provided by the OpenCV library [30], which uses Levenberg-Marquardt optimization [60].

We use the warped segmented image as the blueprint of our 2D navigable area model, placed in the 3D environment. We use its four corners as points in our 3D world, which correspond to the corners of the image. The flat navigation mesh is placed on the XZ-plane, therefore, the Z axis of the model coordinates are taken as $image_{height} - v$ from the image and Y is 0. When running the PnP solver, we provide the intrinsic matrix K and assume zero distortions, which would affect the projection. The initial run of the solver is generally inaccurate because of metric rectification and focal length errors. Fig. 6(a) shows a sample output. The axes that represent the projections of the four corners of the model are away from the corners of the image (see Fig. 6(b)).

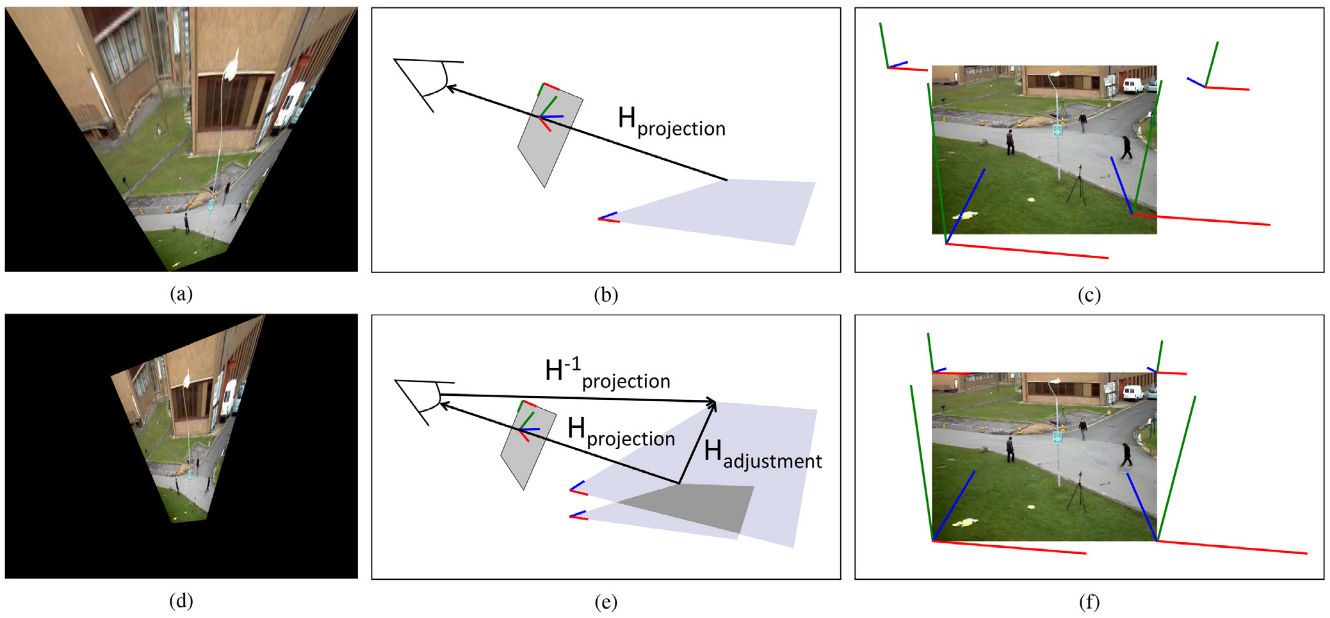


Fig. 6. The model adjustment process starts with an initial solution to plane-to-plane homography using the PnP solution (b) for (a). As the resulting placement (c) is noisy, we multiply $H^{-1}_{projection}$ with image corners (e) to obtain the perfect fitting model (d). In (c) and (f), the RGB lines correspond to X, Y, and Z coordinates in Unity. The model images in (a) and (d) are not in scale with illustrations in (b) and (e).

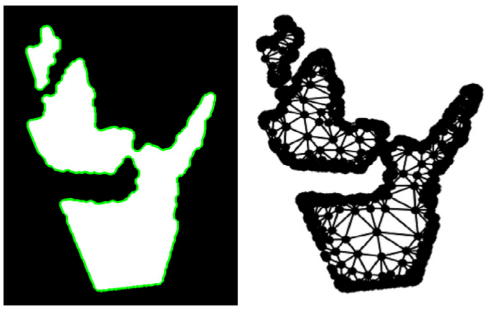


Fig. 7. Example triangulation of the navigable area. We preprocess the model image with erosion and dilation for refinement.

Table 1
The quantitative comparison of various pedestrian trackers. Overall, deep learning approaches can achieve more detections at the expense of more false positives; as seen in lower precision, MOTA, and with higher recall, MOTP than HOG. FPS is frames processed per second. The deep learning approaches have lower FPS because they were run using a CPU. The experiments were performed on a personal computer with Intel®Core^{TM}i7-4500U CPU @1.8 GHz, 8 GB RAM, and NVIDIA 740M.

PETS09-S2L1 (768×576 @15 fps)					
Method	Recall	Precision	MOTA	MOTP	FPS
HOG	81.8	76.8	55.5	0.314	0.5
Inception v2	92.8	47.6	-10.5	0.241	0.114
MobileNet	91.4	63.2	36.8	0.244	0.188
Custom (1280×720 @30 fps)					
Method	Recall	Precision	MOTA	MOTP	FPS
HOG	18.1	41.4	−7.8	0.400	0.243
Inception v2	42.9	31.8	−50.2	0.343	0.075
MobileNet	30	28	−48	0.334	0.115

Assume that there is a projection matrix P that maps the 3D corners of the model to their projections on the image plane (Eq. (8)). 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, we remove the Y component of the 3D position vector. This turns our problem into plane-to-plane homography.

Table 2
Dice scores of the segmentation results (higher is better) for PETS09-S2L1 [44], Town Centre [62], MOT16-04[31] and our custom video. “No. frames” is the number of frames used for segmentation update, “Threshold” is density ratio to filter rarely navigated areas. High dice scores with lower than 50 segmentation updates indicate that our approach produce accurate navigable regions with threshold as 0.3.

		Threshold			
Scene	No. frames	0.0	0.3	0.7	0.95
PETS09-S2L1	40	0.887	0.895	0.711	0.711
Town Centre	46	0.945	0.945	0.945	0.912
MOT16-04	35	0.912	0.912	0.912	0.912
Custom video	38	0.833	0.833	0.833	0.833

We use the PnP solver to obtain the projection of the corners of the initial model on the image plane. We calculate $H_{projection}$ between the corners of the model in the model plane and the projection in the image plane using four-point correspondence (Eq. (9)). Then by applying $H^{-1}_{projection}$ to the original image corners, we obtain the corners for the ideal model on the XZ-plane (Eq. (10)). Because we now know the ideal model, we use homography to warp our existing model corners to its corners (Eq. (11)). The homography matrix $H_{adjustment}$ is found using 4 points correspondence between each models' corners on the XZ-plane. After adjusting the model, we estimate the camera pose again using an iterative PnP solution. To keep the corners of the adjusted model inside the image, we minimize the initial model and apply similarity transformations to the final model. Fig. 6(c) and (d) summarize this correction process. We use the result of the PnP solution together with the image frame properties (e.g., resolution, center) in Unity. We use the field of view calculated from the focal length as the vertical FoV.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{projection} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}_{initial} \tag{8}$$

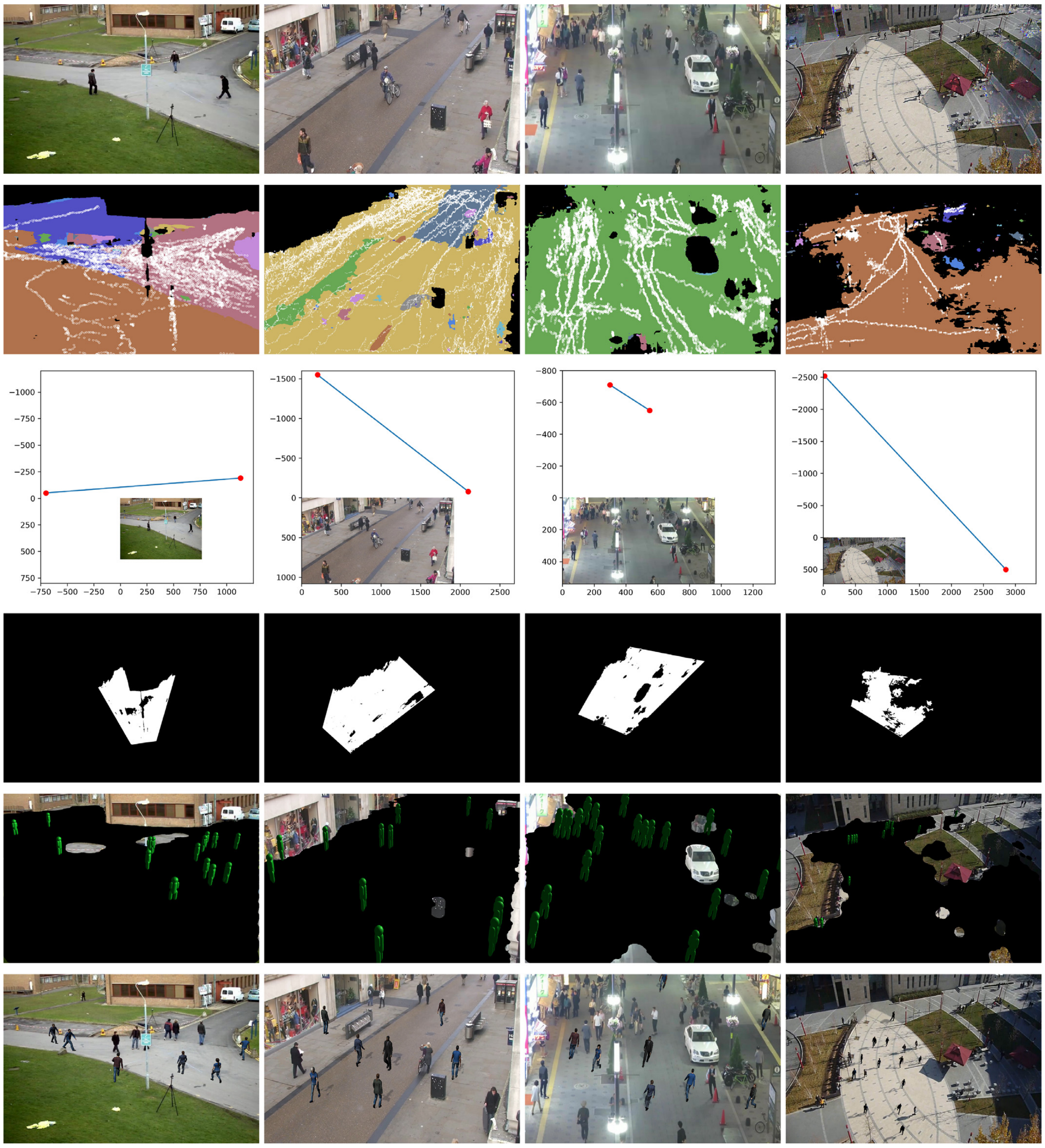


Fig. 8. The example reconstruction of the navigable area model for each video. From top to bottom: the original frame, navigable areas, calculated horizons, the final model blueprint, detected pedestrians on top of the flat navigation mesh (rendered in black) projected on top of the input video, and the output with real pedestrians and virtual agents. From left to right: PETS09-S2L1[44], Town Centre [62], MOT16-04 [31] and our custom video. In some videos, the postures of the projected pedestrians do not perfectly match the pedestrians in the video because of the small errors in the horizon and focal length calculations. We reconstruct the navigable area and place it on the 3D scene successfully even in the presence of such errors.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{projection} = H_{projection} \begin{bmatrix} X \\ Z \\ 1 \end{bmatrix}_{initial} \tag{9}$$

$$H^{-1}_{projection} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_{corners} = \begin{bmatrix} X \\ Z \\ 1 \end{bmatrix}_{ideal} \tag{10}$$

$$\begin{bmatrix} X \\ 0 \\ Z \\ 1 \end{bmatrix}_{ideal} = H_{adjustment} \begin{bmatrix} X \\ 0 \\ Z \\ 1 \end{bmatrix}_{initial} \tag{11}$$

4.4. Mesh construction

After determining the navigable regions in the video frame and obtaining their frontal view blueprint, we need to convert the blueprint to a mesh on which agents can navigate. To this end, we implemented a mesh generator that converts the given binary image into a 2D mesh. White areas in this binary image represent navigable regions. We assume that the area is flat; there are no stairs or any elevation change in the environment.

We apply dilation and erosion operations on the binary image to eliminate possible noise, tiny clusters, and holes. Then we find contours using OpenCV. We use the tree hierarchy of OpenCV that enumerates each contour from outer to inner. To convert this contour hierarchy of border vertices and edges into a mesh, we use the *Triangle* [61] framework. This framework uses the notion of a "triangle-eating virus" to process the holes, which removes the triangulation from its initial point to the closest segment it reaches. The planting location of the virus needs to be within the whole area. 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, which will be the initial point of the virus. Fig. 7 shows a generated mesh example. After the triangulation, the resulting mesh is exported to Unity.

5. Evaluation

We test our framework on various stationary surveillance-like videos including PETS09-S2L1 [44], Town Centre [62], MOT16-04 [31], and a custom video. Fig. 8 includes the horizon, extracted navigable areas, their placement into the 3D scene with dummy agents, and the final output with virtual agents for each test video.

Table 1 shows a quantitative comparison of different pedestrian trackers for PETS09-S2L1 and our custom video. *Recall* is the percentage of identified pedestrians overall in the video, whereas *precision* is the percentage of correct detections overall. MOTP and MOTA represent multi-object precision and accuracy, respectively [63]. MOTP measures the mean dissimilarity between the correct detections and the ground truth, where the smaller is better. MOTA focuses on mismatch errors that occur when a single object is identified multiple times as different objects. It can result in negative values when the detection errors are higher than the number of pedestrians.

The results indicate that deep learning-based methods achieve detections that capture a high number of pedestrians but some of them are false positives. This would populate the navigable region with more pedestrians than there are in the video; limiting the navigation of augmented agents. However, for augmented reality applications, missing pedestrians in the video is a more important problem than false positives; therefore, deep learning-based approaches are preferable.

To evaluate the performance of our reconstruction method, we measure the similarity of the automatically determined navigable areas in comparison to the ground truth. We report this similarity in terms of the dice score using different parameter configurations. The dice score represents the overlapping pixel area percentage between the ground truth and the predicted regions, in the range [0.0, 1.0]. We manually craft segmentation maps of each video to use as the ground truth. We test two available network architectures for semantic segmentation: Xception [48,49] and

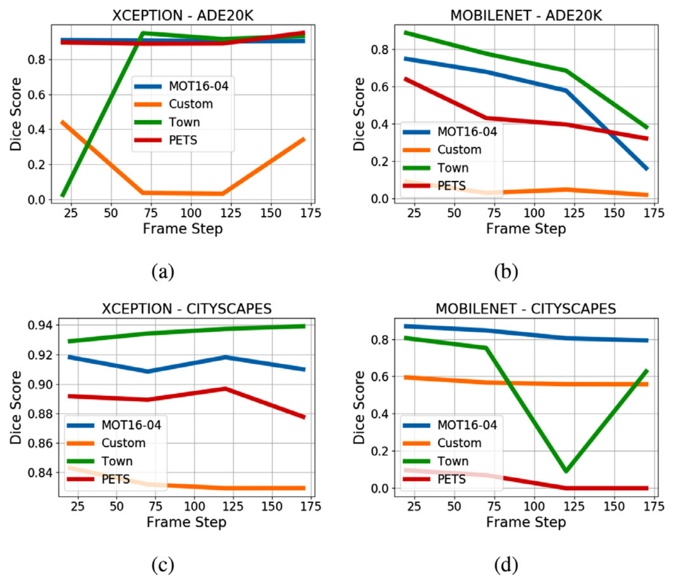


Fig. 9. Segmentation performance in terms of dice scores (higher is better) for each video. Overall, the performance is more stable with Xception [48,49] / Cityspaces [54] combination while decreasing segmentation frequency. PETS video has problems with MobilenetV2 [47], especially when using Cityspaces [54]. We obtain the best results for our custom video when using Xception/Cityspaces.

MobileNetV2 [47], and compare two datasets for the training: ADE20K [52,53] and Cityspaces [54]. In each combination, we also try different frame steps (skipped frame count between each segmented frame).

Fig. 9 shows the dice score of different network-training set combinations for each video. Throughout the experiments, we kept the shadows and fixed the pedestrian navigation density threshold to 0.33. The results show that the Xception/Cityspaces combination yields the most stable results in terms of performance. Overall, the dice score decreases as the frame step increases, especially when we use Mobilenet/ADE20K. We did not observe significant changes in the dice score when we increase the pedestrian navigation density threshold from 0.3 to 0.7. We think this is related to the crowd density of the scenes. However, this experiment shows that the segmentation frequency is an important parameter that influences performance.

Although we can use all frames of the input video, this increases the running time significantly. We show that we can skip some frames of the input without losing accuracy. For example, when using Xception/ADE20K, we can segment one frame per 100 frames of PETS and maintain accuracy. This decreases the running time of the network by 100 fold. If the frame step is too high, then the network does not have enough information on the regions occluded by pedestrians. For example, the segmented frames could include different pedestrians at the same position at different times, thus the network cannot capture a clear view of the occluded region. This effect is more obvious in crowded scenes. Additionally, in certain cases like Town in Xception/ADE20K, using a lower frame step results in lower accuracy. We believe this is caused by the network capturing the noise rather than focusing on the overall scene. This also justifies skipping some frames.

Table 2 shows the quantitative segmentation results for the test videos. For these results, we process all videos using the Xception network based on ADE20K dataset labels, except for the custom video, which is based on Cityscapes labels. A dice score close to 1.0 indicates high similarity. We tested different threshold values for each video, where 0.0 means only pre-defined navigable labels are used, not the pedestrian trajectory data. Our custom video

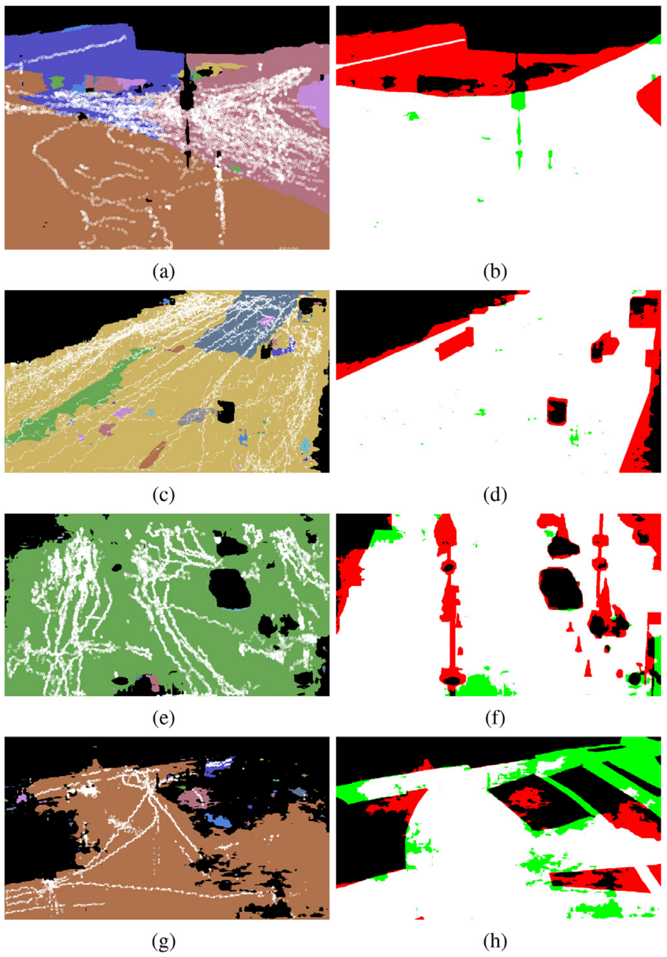


Fig. 10. Qualitative comparison between example predicted and ground truth navigable regions. Non-white pixels, red and green, are only navigable in prediction and ground truth respectively. From top to bottom: PETS09-S2L1[44], Town Centre [62], MOT16-04 [31] and our custom video. The segmentation process obtains satisfactory results as long as the navigation density of the pedestrians is high enough, in terms of the user-defined threshold. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

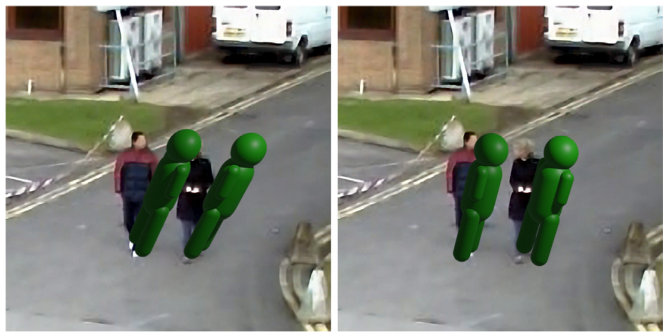


Fig. 11. Differences between the positions and orientations of impostors and pedestrians in the video. A camera with a focal length smaller than expected will be placed closer to the area, resulting in misplaced impostors (left). The impostors are better positioned and oriented when the focal length approximation is more accurate (right).

Table 3
RMSE values (lower is better) and failed projection rates (lower is better) along with basic camera configuration for each video. P_{height} denotes the pedestrian height, C_{height} denotes the camera height, and NC_{height} is the normalized camera height obtained by normalizing the pedestrian height to 1.65 meters and then adjusting the camera height accordingly. Except for low FoV cases (MOT16-04), we obtain relatively good results in terms of *RMSE*. The custom video suffers from a high failed projection rate due to its poor area segmentation result.

Metric / Data	PETS	Town	MOT	Custom
RMSE	0.051	0.036	1.45	0.110
Failed (%)	0.006	5.99	10.09	52.60
FoV	34.95	24.8	9.83	34.05
P_{height}	24	79	46	26
C_{height}	131.45	437.57	569.09	457.83
NC_{height}	9.04	9.14	20,41	29.06

poses a difficult challenge because of its complex structure and shadows. To improve the performance in this video, we remove shadows and increase the segmentation update frequency. When we increase the threshold, we can see an immediate improvement in PETS video, which is most likely due to getting rid of noisy areas. However, increasing the threshold further starts decreasing the dice score because navigable areas can be lost in the process. This does not apply to MOT16-04 and our custom video, however, as they contain a large singular navigable region rather than many small chunks (cf. Fig. 10).

Fig. 10 provides a qualitative comparison between predicted navigable areas and ground truth (manually annotated). Except for our custom video where the navigable region is distant from the camera, we successfully segment most of the navigable regions and surpass small occlusions (such as poles and traffic cones).

Table 3 shows the performance of our reconstruction method in terms of Root-Mean-Square-Error (*RMSE*) [17] following Eq. (12), and failed projection rates.

$$RMSE = \sum_{gt} \left(\frac{dist(p_h, p_h^{gt})}{dist(p_f, p_h^{gt})} \right)^2, \tag{12}$$

where p_h and p_f are the 2D head and foot positions, respectively, for pedestrians on the image plane as we project to the navigable region in the 3D environment. p_h^{gt} are the ground truth head positions of pedestrians on the image plane. RMSE represents the accuracy of our camera configuration estimation in terms of the match between the postures of real pedestrians and corresponding dummy agents in the 3D scene. A low error shows that the height and posture of dummy agents are very close to that of real pedestrians. We sum *RMSE* for each pedestrian in the ground truth detection (denoted as *gt*) that we can successfully project onto the 2D navigable map. Failed projections do not contribute to the calculation of *RMSE*. For the pedestrian height, we take the first height (in Unity units) determined while the simulation is running.

Except for MOT16-04, the *RMSE* rates are as low as 0.036. The higher *RMSE* rate for MOT16-04 is due to its low Field of View (FoV) angle, which causes the camera to be positioned relatively high. In such failure cases, the focal length is very different from the expected value. Even though the focal length does not affect perspective correction, it becomes important during camera placement. Camera placement takes into account the calibration process 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 its actual counterpart in the real world, and the inverse applies to a longer focal length. This causes impostors (dummy models used for collision detection and avoidance) to be misplaced in the 3D scene (cf. Fig. 11 for an example from PETS09-S2L1 [44]). A good focal length estimation is as important as a decent horizon orientation to reduce projection errors.

Table 3 also shows the calculated camera height relative to the ground (y=0). For comparison, we normalize the calculated height to the average human height of 1.65 m. Table 4 compares our results and the ground truth configuration from Town Centre [62]. We do not have the ground truth configurations of the other videos. The results show that we can successfully project our

Table 4
The comparison of the calculated and ground truth camera internal and external parameters for Town Center [62] video.

Ground Truth		Calculated	
Focal length	Camera height	Focal length	Camera height
2696.30	12.39	2455.83	9.14

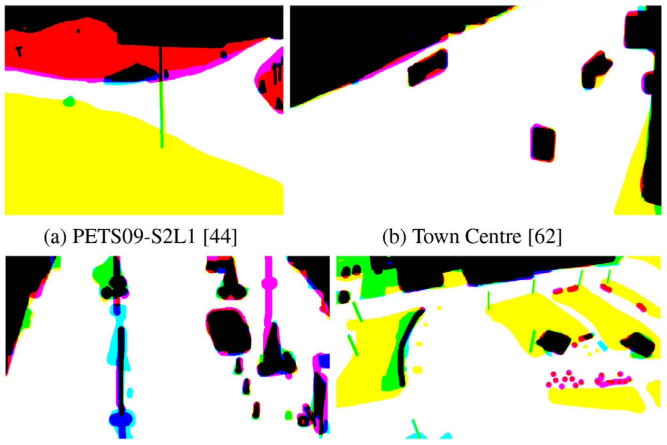


Fig. 12. The union of three of the manually created navigable region maps for each input video by three different users. Each color channel in these RGB images represents the navigable areas determined by one user. White corresponds to the common navigable regions and black corresponds to the common impassable regions.

models with a very small error because our camera view is as if the original camera is put closer to the scene with a lower zoom (higher FoV).

Overall, we can detect a horizon even under complex scenarios where there are many pedestrians and noisy image features, for example, MOT16-04. We can fit the corrected navigable area model onto the 3D scene for the input video with minimal error in terms of RMSE. Even in cases where the detected horizons have errors, the proposed navigable area correction method successfully rectifies the navigable region and matches the camera orientation.

Table 5
Average (Avg.) and Standard Deviation (SD) for the dice scores and time measurements (in seconds) of the manual approach in the first preliminary user experiment with 17 participants for each scene. We compare the average user dice scores to the performance of our implementation.

	Manual				Automatic
Scene	Dice Avg.	Dice SD.	Time Avg.	Time SD	Dice
PETS09-S2L1	0.835	0.140	202.6	162.4	0.895
Town Centre	0.963	0.017	230.7	181.7	0.945
MOT16-04	0.950	0.017	309.6	344.8	0.912
Custom video	0.828	0.045	358.5	367.7	0.833

Table 6
Average time (in seconds) for the users to finish modelling the navigable mesh and position it into the scene with appropriate camera settings. We compare this to the total running time of our system using the settings that yield the best navigable region map accuracy.

Scene	Avg. User Time (s)	System Time
PETS09-S2L1	438	175.67
Town Centre	230	183
MOT16-04	764	93.7
Custom video	805	126.99

With two follow up preliminary user experiments, we compare the results of our system to the manual approach in terms of speed and accuracy. In the first preliminary user feedback, we ask 17 different users (70.6% male and 29.4% female) to paint the impassable regions on top of each input video. Participants are graduate and undergraduate university students with an average age of 23.4 ± 2.7. Participants use an online tool for painting the impassable regions using their personal computers. Three participants use a graphics tablet, and the remaining fourteen participants use a computer mouse. The online tool first displays the instructions and then plays the video on a loop. The participants can pause and resume the video as they desire. The tool starts to keep the time with the first stroke of the participant. Different brush sizes and erasers are available to participants. In the resulting navigable region maps, the black color shows the impassable regions and the white color shows the navigable regions. In Fig. 12, we combine

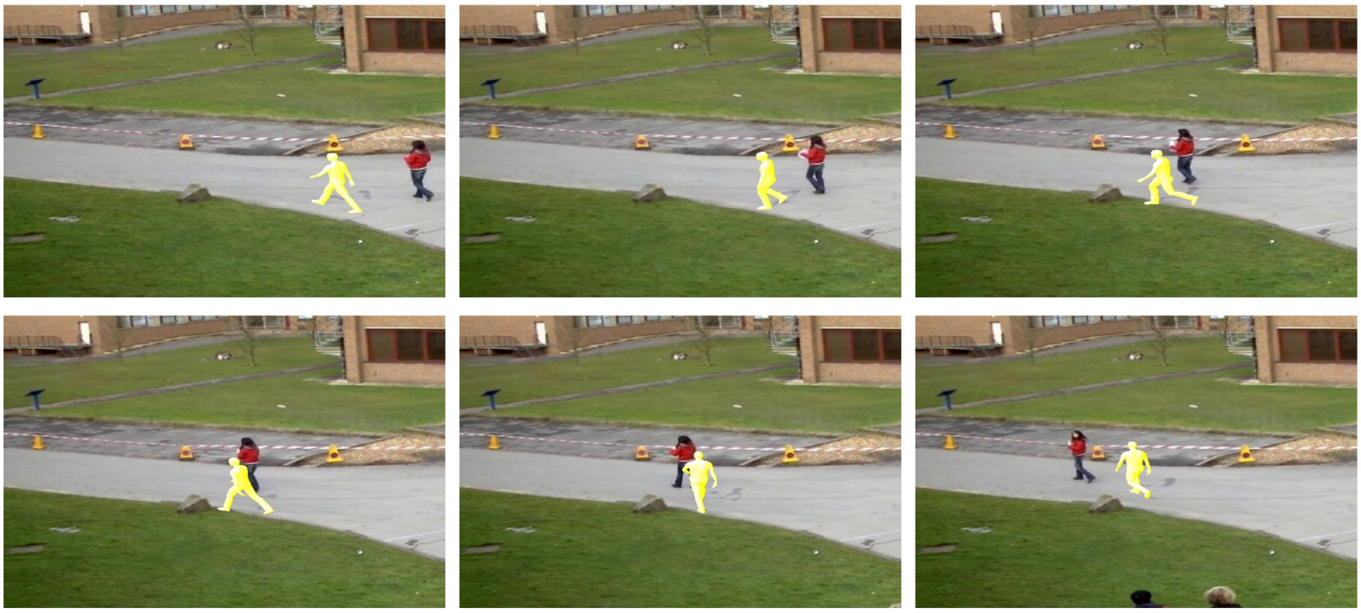


Fig. 13. Zoomed in still frames showing the collision avoidance of the virtual agent (indicated with yellow), when a faster real pedestrian comes from behind, using PETS09-S2L1 [44] video. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

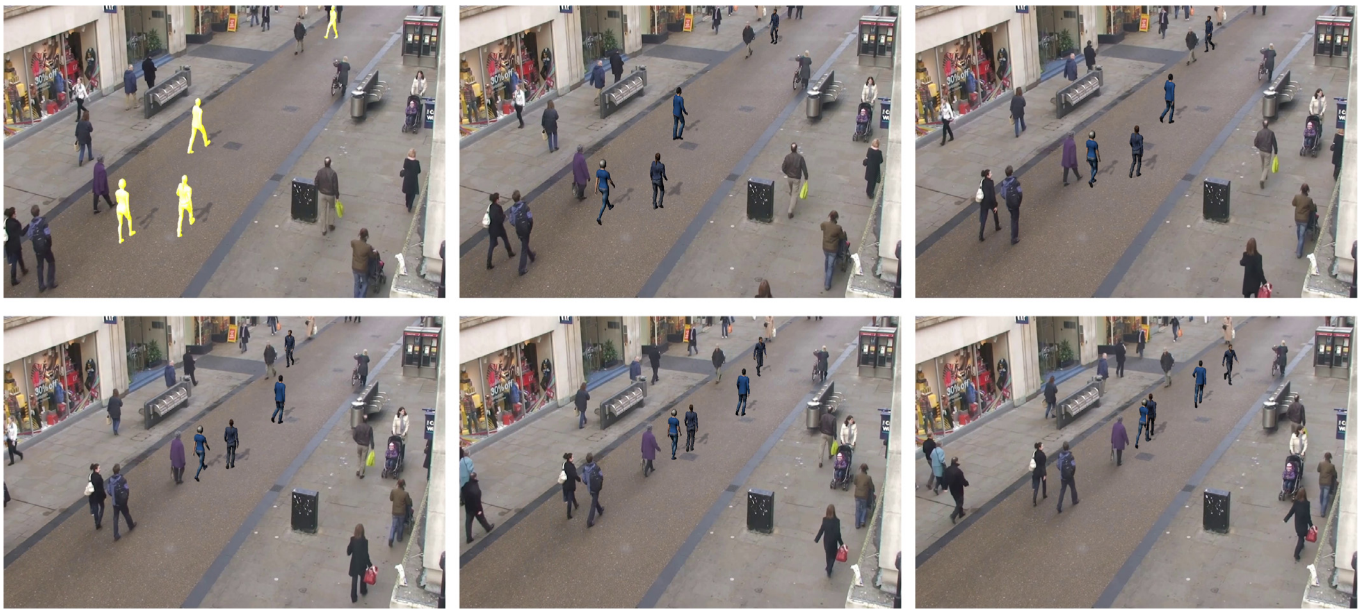


Fig. 14. Still frames showing the virtual agents (indicated with yellow in the first frame) walking on the navigable regions of the environment, while avoiding collision with other virtual agents and real pedestrians, using Town Centre [62] video.) (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

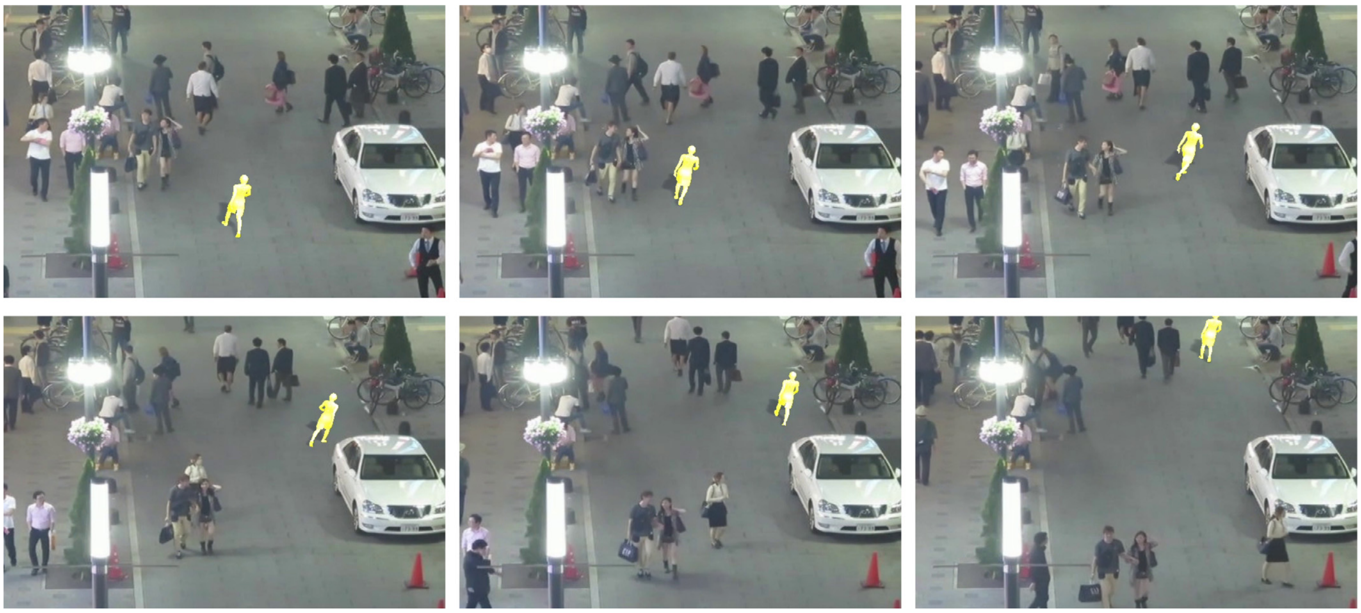


Fig. 15. Zoomed in still frames showing the virtual agent (indicated with yellow) walking on the navigable areas of the environment, with the user-defined destination on the top right, avoiding collision with the real pedestrians, using MOT16-04 [31] video. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the maps created by a subset of three users into RGB images such that each color channel comes from one user.

In Fig. 12(b), we see that users mostly agree on the navigable regions. We believe this is because in this scene impassable regions are more obvious. For example, in Fig. 12(a) and (d), there is high variance. While some users regard grass areas and construction sites as impassable some users do not. This is a result of the assumptions of the users; in the user study, we do not specify whether any of these regions should be regarded as navigable or not. Some users determine the grass regions as navigable based on pedestrians that walk on such areas. Dice scores calculated using these user-defined navigable regions are given in Table 5. We see that lack of consistency results in poor accuracy while in simpler scenes such as Town Centre, the accuracy of the manual approach is similar to our results. However, we believe that consistency is important if multiple scenes are analyzed by different users, thus using an automatic approach is necessary.

We include frames from our crowd simulation examples in Figs. 13–16. In Fig. 13, we focus on a virtual agent that avoids collision with a fast-moving pedestrian that comes from behind in PETS09-S2L1 [44]. The virtual agent moves slightly out of its path giving way to the real pedestrian. In Fig. 15, the virtual agent updates its track to avoid multiple real pedestrians in MOT16-04 [31]. Figs. 14 and 16 show virtual crowds of different densities blending in the environment of the input video, which is made possible by the correct positioning of the 3D camera and the navigation

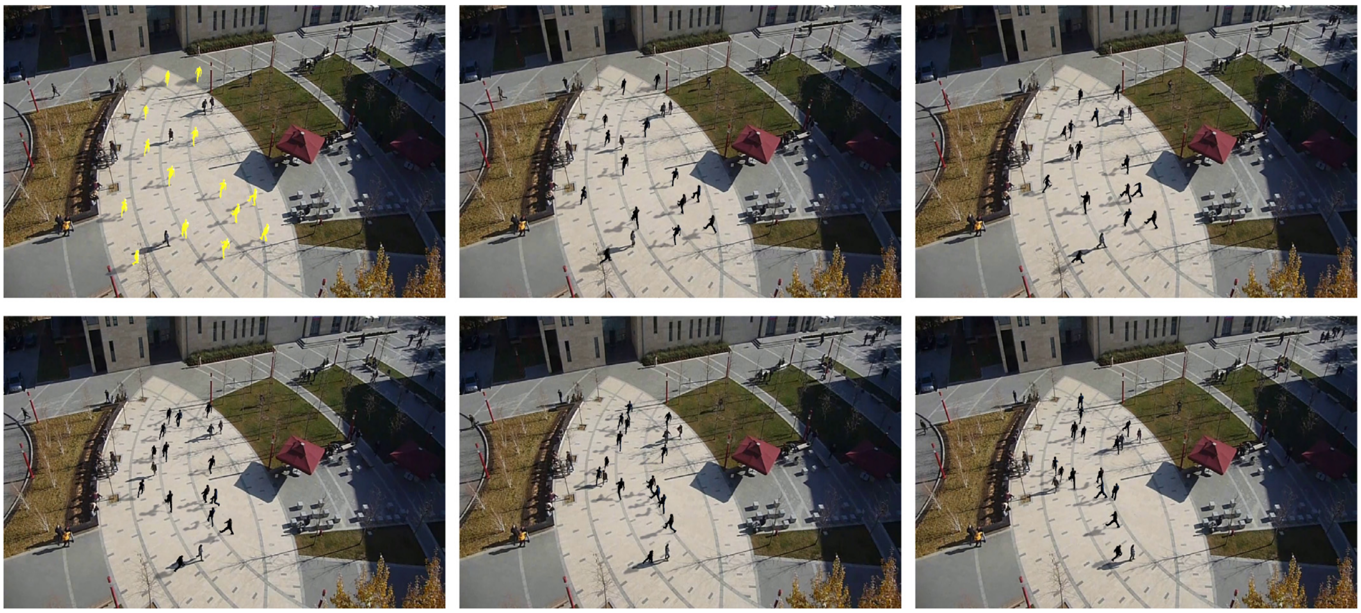


Fig. 16. Still frames showing the virtual agents (indicated with yellow in the first frame) navigating in the environment of our custom video. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

mesh. Virtual agents in these videos also avoid collision with real pedestrians.

In the second preliminary user experiment, we ask three users who participated in the first experiment to model the flat mesh that corresponds to the navigable regions of the scene in the given videos. These three users (2 male and 1 female, average age: 24.6 ± 3.7) are the ones that indicate a high skill level in 3D modeling. We also ask the user to position this flat mesh into the 3D scene with appropriate camera settings. Participants are free to use any 3D modeling software with which they are comfortable. They can view the video and the navigable region map they labeled in the first experiment while modeling and positioning. Table 6 shows that our system generates the resulting flat mesh and places it into the scene with appropriate camera configurations much faster than the average user time. The system time includes segmentation, rectification, mesh generation, and simulation loading. The system time includes the user's line feature parameter adjustments in rectification to obtain the best result in cases where pedestrian trajectories are noisy.

6. Conclusion

We introduce an open-source augmented crowd simulation system that utilizes automatic determination of navigable regions in surveillance-like videos. The GitHub project including the repositories that contain the source codes of the proposed system is located at https://github.com/users/YalimD/projects/2. We combine existing techniques of semantic segmentation and pedestrian detection for accurate determination of the navigable regions. We compare our results with the ground truth and manually labeled navigable regions. As opposed to the manual approach, we show that our solution generates more consistent navigable region maps. Although the users can identify the objects of the scene better, their decision about which segments are navigable is subjective and therefore is not consistent among different users.

As an example use case scenario, we integrate the resulting navigable regions in augmented crowd simulations. To this end, we generate a flat navigation mesh using the navigable region map and simulate virtual agents on its surface. Thanks to the automatically calibrated scene camera, virtual agents appear to walk inside the navigable regions of the scene while avoiding collision with real pedestrians.

In terms of generating the corresponding navigable mesh from raw footage, our solution works much faster than the average user. The difficulty of the manual approach is modeling a flat mesh purely based on single view footage with perspective. The user needs to imagine the view of the scene from above and model the corresponding flat mesh. Additionally, for the augmentation task, the 3D scene camera should imitate the original camera of the input footage. Users need to try many configurations to determine the appropriate camera settings while our system works faster. As there is no ground truth available for the 3D scenes, our evaluation focuses on the generated navigable maps and for the final output, we present visual results. Since we make use of known techniques for 3D reconstruction, we expect the navigable meshes to be accurate since the input navigable regions are shown to be accurate.

The use case of the resulting system applies to filling real environments with virtual characters as in movies. For example, in a wide shot where the main characters are acting, we can fill the otherwise unoccupied regions with virtual characters to increase immersion. In this case, it is important for the virtual characters to only exist in the navigable regions of the scene so as not to hurt the realism. Since such virtual characters may walk inside the scene, they need to avoid collision with the main characters that preexist in the input video. With our proposed solution animated characters can be augmented into real-life videos without any manual labeling.

The same scenario also applies to integrating virtual tutors into real-life scenes. For example, let's consider a city guide that uses real-life footage of different environments. To keep the content up to date, we may utilize daily footage and thus there could be minor or major differences in the navigable regions of a particular scene (i.e., cars or construction may block certain regions). In this case, our automatic approach can generate the current navigable mesh daily. We can use the same origin and destination for the virtual tutor and it will navigate according to the current navigable regions and pedestrians without a need for manual labor.

The resulting navigation mesh can also be used for navigating robot agents in real environments using live surveillance feed. For example, a drone camera can record short videos at specific locations to reconstruct the navigable regions for the robot to navigate. While the navigable regions are detected using multiple previous frames, the current dynamic obstacles can be determined using the current frame of the live feed. For this example, manual labeling of the regions is not feasible as the environments could be previously unseen and require fast determination of navigable regions. This also justifies our use of segmentation on a limited number of frames. As our results indicate, we do not need all of the frames of the input video for accurate reconstruction and therefore we can get the same accuracy much faster.

Our method can function in arbitrary scenes with minimal geometric information. Even in cases where the detected horizon and the calculated focal length have errors, the navigable area correction step compensates the distortion to a level so that the final model matches the image corners after projection.

The limitations of our work, including the possible further research areas, can be listed as follows:

- Noisy data in the input video can make it hard to find a valid horizon, which may create cases in which the navigable area correction process cannot handle. Furthermore, our horizon calculation algorithm is sensitive to parameters such as the pedestrian posture sample rate, and scene line configuration.
- Our rectification algorithm cannot handle cases where the horizon is visible in the image. This is because the 3D model points are expected to match with the 2D image points. However, the points above the horizon do not satisfy this constraint. In this case, it is necessary to crop the input image so that all pixels lie below the horizon. Afterward, we should calculate the camera configuration accordingly so that the corners of the cropped image can match the corners of the cropped model.
- Our navigable area extraction method takes only the trajectories of pedestrians in the video into account when determining navigability, which might limit the area of navigable regions.
- We do not consider the illumination conditions of the input footage, and the possible occlusion of real pedestrians by virtual agents. We render each virtual agent in front of the video and the real pedestrians, even if the agent is behind the pedestrian in the 3D scene.
- For projected pedestrians to be able to occlude virtual agents, we can utilize the foreground pixels of background subtraction and follow a "billboarding" approach: use each pixel cluster as a textured plane at its corresponding pedestrian location and use it as an occlusion mask.
- We omit immobile pedestrians in our pedestrian detection stage for increased performance. Idle pedestrians that start moving could cause sudden velocity changes in virtual agents that appear unnatural. Pedestrian detection could work on the whole image in exchange for fast preprocessing in videos where idle pedestrians are frequent.

Declaration of Competing Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors are grateful to Lori Russell-Dağ and İpek Sözen for proofreading the manuscript.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2021.01.012.

References

[1] Musse SR, Jung CR, Jacques J, Braun A. Using computer vision to simulate the motion of virtual agents. Comput Anim Virtual Worlds 2007;18(2):83–93.
[2] Lerner A, Chrysanthou Y, Lischinski D. Crowds by example. Comput Graph Forum 2007;26(3):655–64.
[3] Kim S, Bera A, Best A, Chabra R, Manocha D. Interactive and adaptive data-driven crowd simulation. In: Proceedings of IEEE virtual reality. VR '16;. IEEE; 2016. 29–38.
[4] Jablonski K, Argyriou V, Greenhill D, Velastin SA. Evaluation framework for crowd behaviour simulation and analysis based on real videos and scene reconstruction. In: Proceedings of the 6th Latin-American conference on networked and electronic media. LACNEM '15;. IET; 2015. 1–6.
[5] Amirian J, van Toll W, Hayet JB, Pettré J. Data-driven crowd simulation with generative adversarial networks. In: Proceedings of the 32nd international conference on computer animation and social agents. CASA 19;. New York, NY, USA: Association for Computing Machinery. 7–10.
[6] Amirian J, Hayet JB, Pettre J. Social ways: learning multi-modal distributions of pedestrian trajectories with GANs. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops. CVPRW '19;; 2019b. p. 2964–72.
[7] Bera A, Kim S, Manocha D. Online parameter learning for data-driven crowd simulation and content generation. Comput Graph 2016;55:68–79.
[8] Zheng F, Li H. ARCrowd-a tangible interface for interactive crowd simulation. In: Proceedings of the 16th international conference on intelligent user interfaces. IUI '11;. ACM; 2011. 427–430.
[9] Baiget P, Fernández C, Roca X, Gonzàlez J. Generation of augmented video sequences combining behavioral animation and multi-object tracking. Comput Anim Virtual Worlds 2009;20(4):473–89.
[10] Olivier AH, Bruneau J, Kulpa R, Pettré J. Walking with virtual people: Evaluation of locomotion interfaces in dynamic environments. IEEE Trans Vis ComputGraph 2018;24(7):2251–63.
[11] Rivalcoba JI, De Gyves O, Rudomin I, Pelechano Gómez N. Coupling camera-tracked humans with a simulated virtual crowd. In: Proceedings of the 9th international conference on computer graphics theory and applications. GRAPP '14;. SciTePress; 2014. 312–321.
[12] Zhang Y, Pettre J, Ondřej J, Qin X, Peng Q, Donikian S. Online inserting virtual characters into dynamic video scenes. Comput Anim Virtual Worlds 2011;22(6):499–510.
[13] Doğan Y, Demirci S, Güdükbay U, Dibeklioğlu H. Augmentation of virtual agents in real crowd videos. Signal Image Video Process 2019;13(4):643–650.
[14] Li B, Peng K, Ying X, Zha H. Vanishing point detection using cascaded 1D Hough Transform from single images. Pattern Recognit Lett 2012;33(1):1–8.
[15] Zhai M, Workman S, Jacobs N. 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; 2016. p. 5657–5665.
[16] Trocoli T, Oliveira L. Using the scene to calibrate the camera. In: Proceedings of the 29th SIBGRAPI conference on graphics, patterns and images. SIBGRAPI'16. IEEE; 2016. 455–461.
[17] Liu J, Collins RT, Liu Y. Surveillance camera autocalibration based on pedestrian height distributions. In: British machine vision conference. In: BMVC '11, 2; 2011. p. 1–11.
[18] Brouwers GM, Zwemer MH, Wijnhoven RG, de With P. Automatic calibration of stationary surveillance cameras in the wild. In: European conference on computer vision. ECCV '16;. Springer; 2016. 743–759.
[19] Jung J, Yoon I, Lee S, Paik J. Object detection and tracking-based camera calibration for normalized human height estimation. J Sens 2016;2016. Article no. 8347841, 9 pages
[20] Liebowitz D, Zisserman A. Metric rectification for perspective images of planes. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition. CVPR '98;. IEEE; 1998. 482.
[21] Liebowitz D, Criminisi A, Zisserman A. Creating architectural models from images. Comput Graph Forum 1999;18(3):39–50.
[22] Bose B, Grimson E. 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; 2003. p. 94–101.
[23] Chaudhury K, DiVerdi S, Ioffe S. Auto-rectification of user photos. In: IEEE international conference on image processing. (ICIP '14). IEEE; 2014. 3479–3483.
[24] Bulbul A, Dahyot R. Populating virtual cities using social media. Comput Anim Virtual Worlds 2017;28(5). Article no. e1742, 10 pages.
[25] Iizuka S, Kanamori Y, Mitani J, Fukui Y. Efficiently modeling 3D scenes from a single image. IEEE Comput Graph Appl 2012;32(6):18–25.
[26] Zhang G, Qin X, An X, Chen W, Bao H. As-consistent-as-possible compositing of virtual objects and video sequences. Comput Anim Virtual Worlds 2006;17(3-4):305–14.
[27] Hoiem D, Efros AA, Hebert M. Automatic photo pop-up. ACM Trans Graph 2005;24(3):577–84.
[28] Saxena A, Sun M, Ng AY. Make3D: learning 3D scene structure from a single still image. IEEE Trans Pattern Anal MachIntell 2009;31(5):824–40.
[29] Team U.. Unity. Accessed 07 Sep. 2020a. Available at http://unity3d.com/.
[30] Team O.. OpenCV (open source computer vision library). Accessed 07 Sep. 2020b. Available at http://opencv.org.
[31] Milan A, Leal-TaixéL, Reid ID, Roth S, Schindler K. MOT16: a benchmark for multi-object tracking. CoRR 2016. http://arxiv.org/abs/1603.00831.

[32] Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision. In: Proceedings of the 7th international joint conference on artificial intelligence. In: IJCAI '81, 2; 1981. p. 674–9. Vancouver, BC, Canada.
[33] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: Proceedings of the IEEE computer society conference on computer vision and pattern recognition. In: CVPR '05, 1. IEEE; 2005. 886–893.
[34] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: single shot multibox detector. In: European conference on computer vision. ECCV '16. Springer; 2016. 21–37.
[35] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the ieee conference on computer vision and pattern recognition. CVPR '15; 2015. p. 1–9.
[36] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR '16; 2016. p. 2818–26.
[37] Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, et al. MobileNets: efficient convolutional neural networks for mobile vision applications. CoRR 2017. http://arxiv.org/abs/1704.04861.
[38] Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE Conference on computer vision and pattern recognition. CVPR '17; 2017. p. 3296–305.
[39] Kurtaev D.. OpenCV tensorflow object detection API. Accessed 07 Sep. 2020. Available at https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API.
[40] Zivkovic Z, van der HF. Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognit Lett 2006;27(7):773–80.
[41] Welch G., Bishop G.. An introduction to the Kalman filter. 1995. Tech. Rep.,Chapel Hill, NC, USA, University of North Carolina at Chapel Hill.
[42] Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation. In: Proceedings of the IEEE international conference on robotics and automation. ICRA '08. IEEE; 2008. 1928–1935.
[43] Van Den Berg J, Guy SJ, Lin M, Manocha D. Reciprocal n-body collision avoidance. In: Robotics research. Springer; 2011. 3–19.
[44] Ferryman J, Shahrokni A. Pets2009: dataset and challenge. In: Proceedings of the twelfth IEEE international workshop on performance evaluation of tracking and surveillance. IEEE; 2009. 1–6.
[45] Murali S, Govindan V. Shadow detection and removal from a single image using LAB color space. Cybern Inf Technol 2013;13(1):95–103.
[46] Murali S, Govindan V. Removal of shadows from a single image. In: Proceedings of first international conference on futuristic trends in computer science and engineering, 4; 2006. p. 111–14.
[47] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC. MobileNetV2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2018. p. 4510–20.
[48] Chollet F. Xception: deep learning with depthwise separable convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR '17; 2017. p. 1251–8.
[49] Dai J, Qi H, Xiong Y, Li Y, Zhang G, Hu H, et al. Deformable convolutional networks. CoRR 2017. http://arxiv.org/abs/1703.06211 in "ICCV COCO Challenge Workshop".
[50] Chen L, Papandreou G, Schroff F, Adam H. Rethinking atrous convolution for semantic image segmentation. CoRR 2017. http://arxiv.org/abs/1706.05587.
[51] Chen LC, Zhu Y, Papandreou G, Schroff F, Adam H. Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Ferrari V, Hebert M, Sminchisescu C, Weiss Y, editors. Proceedings of the 15th European conference computer vision. ECCV '18, Lecture Notes in Computer Science, 11211. Cham: Springer International Publishing; 2018. 833–851.
[52] Zhou B, Zhao H, Puig X, Fidler S, Barriuso A, Torralba A. Scene parsing through ADE20K dataset. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR '17; 2017. p. 5122–30.
[53] Zhou B, Zhao H, Puig X, Xiao T, Fidler S, Barriuso A, et al. Semantic understanding of scenes through the ADE20K dataset. Int J Comput Vis 2019;127(3):302–21.
[54] Cordts M, Omran M, Ramos S, Rehfeld T, Enzweiler M, Benenson R, et al. The CityScapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR '16; 2016. p. 3213–23.
[55] Lv F, Zhao T, Nevatia R. Camera calibration from video of a walking human. IEEE Trans Pattern Anal MachIntell 2006;28(9):1513–18.
[56] Fischler MA, Bolles RC. Random Sample Consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 1981;24(6):381–95.
[57] Hartley R, Zisserman A. Multiple view geometry in computer vision. Cambridge University Press; 2003.
[58] Semple J, Kneebone G. Algebraic projective geometry. Oxford, UK: Oxford University Press; 1979.
[59] Chilamkurthy S.. Github: automated rectification of image. 2016, Accessed 6 Sep. 2020. Available at https://github.com/chsasank/Image-Rectification.
[60] Moré JJ. The Levenberg-Marquardt algorithm: implementation and theory. In: Watson G, editor. Numerical analysis. Lecture Notes in Mathematics, 230. Berlin, Heidelberg: Springer; 1978. 105–116.
[61] Shewchuk J. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. Appl Comput Geom Towards Geom Eng 1996:203–22.
[62] Benfold B, Reid ID. Guiding visual surveillance by tracking human attention. In: Proceedings of the British machine vision conference. BMVC '09; 2009. p. 1–11.
[63] Leal-Taixé L, Milan A, Reid ID, Roth S, Schindler K. MOTChallenge 2015: towards a benchmark for multi-target tracking. CoRR 2015. Accessed 28 Nov. 2018; available at http://arxiv.org/abs/1504.01942.
[64] team T.M.. Makehuman 1.2.0. 2020. http://www.makehumancommunity.org/ Accessed: 2020-12-8.
[65] van den Berg, J., Guy, S. J., Snape, J. Lin, M. C., Manocha, D., RVO2 library: reciprocal collision avoidance for real-time multi-agent simulation. http://gamma.cs.unc.edu/RVO2/ Accessed: 2020-12-12.