Automatic Tracking, Segmentation, Removal and Replay of Traffic in Panoramic 360-Degree Videos for Virtual Reality

Master Thesis

Author: Joey Deiman

Utrecht University

Utrecht University Supervision: Ronald Poppe, Remco Veltkamp

Netherlands Aerospace Centre Supervision: Roalt Aalmoes, Henk Lania

May 23rd, 2016

Table of Contents

Abstract

The Netherlands Aerospace Centre (NLR) has a virtual reality setup which allows users to experience an airplane fly-over with realistically simulated sound. They want to move from using static imagery for the surrounding to video. On top of this, they want to be able to have some form of control over the scenario that they have filmed. This thesis describes a method of automatic detection and segmentation of cars in equirectangular panoramic 360-degree videos by first tracking connected components during an online phase, and doing post processing to enhance the tracks. This will allow the NLR to remove cars from their videos, and replay them at specified times to create custom videos from a single source video. A user study and an objective study have been conducted to determine the quality of the method. The user study has shown that while videos with replayed vehicles are of lower quality than the originals, the custom videos are still acceptable to the majority of users. The objective study shows that the overall accuracy of the method (including replayable cars not used in the user study videos) is not acceptable for a large number of cars. However, as long as there are at least a few usable detections, custom videos can still be produced.

1. Introduction

1.1 The VCNS

Figure 1. The Virtual Community Noise Simulator.

The Netherlands Aerospace Centre (NLR), formerly National Aerospace Laboratory, has a virtual reality (VR) setup called the Virtual Community Noise Simulator (VCNS). This is a device that utilizes a head mounted display (HMD, see Figure 1) and advanced audio techniques in order to let users experience an airplane fly-over in VR with realistic 3D sound. The VCNS works by showing a Google Street View image on a sphere with the sky cut out (and replaced by a uniform blue color) around the user, and having a virtual plane fly over. Aside from the airplane noise, a generic background soundtrack with sounds such as chirping birds, talking people or cars driving by is added. The headset's rotation data is used to calculate the correct 3D audio, and to allow the user to look around. The main use of this simulator is to let users realistically experience different kinds of airplane fly-overs, to let them experience how much noise it would make. This can be used to show the difference between different flight procedures, different airplanes and to demonstrate noise caused by adding a new landing strip nearby.

While the airplane audio simulation of the VCNS is of high-enough quality for its goal, Street View images for visuals are limited in terms of realism. They don't move at all, faces and license plates are blurred out and they often contain other artefacts such as a black circle at the bottom of the sphere where the car used to be. As an example, Figure 2 highlights some of the blurring issues.

Figure 2. Blurring issues in Google Streetview. Partially blurred car in the reflection (left), blurred face (center), blurred license plate + blurred rear of the car (right).

The generic background soundtrack (which is necessary to give a proper real-world noise experience) highlights these issues even more. For instance, the sound of a car moving past the user doesn't actually have a visual representation of the car when the user tries to look at the origin of the sound. Some of these issues will become more obvious to users when using the new higher-quality HMDs [19]. The NLR therefore wants to improve the VCNS for a higher fidelity visual experience. The choice has been made to start using 360-degree videos instead of static images. Moving from Street View images to self-shot footage will remove face and license plate blurring and other artefacts introduced by Street View. It also provides the opportunity to film wherever is needed, instead of relying on locations where Street View is present. The most important improvement however is the fact that there are now moving objects, instead of just static ones.

In addition to using just the recorded videos, the NLR would like to control aspects of the video in order to create different scenarios from a single video. This thesis will focus on controlling the visual traffic aspect of the videos, primarily cars. The 360-degree videos are in an equirectangular panoramic format, which means the full rectangular video frame is utilized, with different amount of deformation depending on how close you get to the top or bottom of the video. See section 3 for details.

1.2 Thesis goal

The goal of this thesis to create a system that can detect, segment and track moving objects (focusing on cars for now) from an equirectangular panoramic video. A static camera is assumed. We remove the segmented cars from the video (leaving just a video of the background), and play them back at user specified times to create a new video.

We aim to extract as many cars as possible with as few visual artefacts as possible (Figure 3).

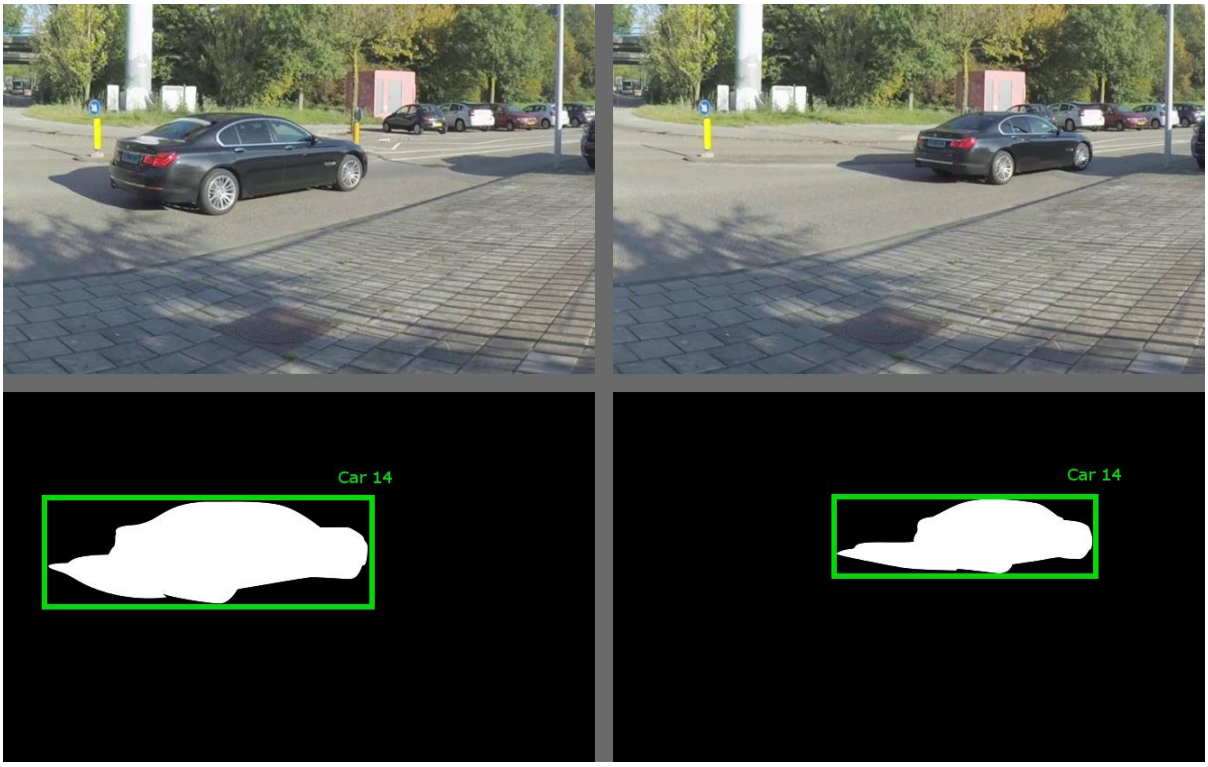

Figure 3. Ideal, non-artifacted car detection and tracking.

More precisely, we want to detect as many pixels of cars (including their shadow) as possible and have as little non-car pixels in our detection as possible. The detected car pixels should be grouped per individual car. We do not distinguish between classes of cars/vehicles (so trucks are detected as well). The objects should be tracked both spatially per frame as well as temporally throughout the video frames. “Detected pixels” are not limited to direct detection, since we will also attempt to fill in pixel of objects that are occluded by another moving object. We want this since when we replay the tracked car, the occluding object is not there anymore, and hence we are missing pixels. While real-time processing is not a requirement, lower computational costs are preferred, since we are working with 4k 60 fps video. Additionally, we aim at reducing the user input as much as possible. This means that we have to be careful with things like drift, since there is no user to correct it mid-tracking. Furthermore, the user does not require any traditional video-editing skills to operate the program. The algorithm must work with arbitrary uncontrolled outdoor scenes, where the objects we aim to detect are in plain sight and unoccluded for at least some amount of time. The scenes should be filmed using a stationary camera.

1.3 Outline

The rest of this thesis is structured as follows: Section 2 will discuss related work regarding moving object detection, object tracking and object segmentation. Section 3 will discuss the proposed method in detail. Section 4 will discuss the results of the objective experiments and user-study. Section 5 will wrap up the thesis with a conclusion and section 6 discusses potential future work.

2. Related work

There are several ways of doing moving object segmentation and tracking in video sequences. Most methods have a similar form. They need some way of determining what pixels belong to the moving foreground, a way to check if the moving pixels form an interesting object and a way of tracking the interesting objects separately over time.

2.1 Detecting moving foreground objects in videos

In many moving object tracking/segmentation algorithms, determining the pixels that form the moving foreground is the first step. The result of such detection is a binary mask per frame which shows what pixels belong to the foreground and which don't. By foreground, we define those objects that have non-repetitive motion, and don't stay in the same area of the scene for extended periods of time. For instance, a car moving from left to right is a foreground object, since it does not repeat its motion constantly in the same area. Instead, it moves throughout the video and eventually disappears. Figure 4 shows an example of foreground detection.

Figure 4. Example of foreground detection. Note the noise in areas where foliage is present. Heavy wind caused erratic motion in the foliage in this particular case.

The methods can be separated into two categories: Those with an explicit background model, and those without such a model.

2.1.1 Explicit background model

We define background as the part of the video where objects are static or semi-static. A tree with leaves moving in the wind would be considered background instead of foreground, since its motion is repetitive and its position therefore is semi-static in the video. A house is an example of a static background object. A background (or foreground) model is a representation of the background (or foreground).

Methods with explicit background models generally utilize a history of frames to decide if a pixel is background or not. These methods attempt to detect pixels that are significantly different from the background. A popular set of methods are those using per-pixel Gaussian Mixture Models (GMMs) [2, 3]. In the basic method, a GMM models the history of values for each separate pixel as a mixture of K Gaussians. Each Gaussian has a weight. Every time a new video frame is processed, the new pixel value is compared with the Gaussians for the pixel's coordinates. If the weighted average correspondence with all the Gaussians is above a certain threshold, the pixel is branded as background, otherwise the pixel is branded foreground. Regardless of the classification of the pixel, the update procedure of the background model is the same. The existing Gaussians' parameters are updated, where the Gaussian closest to the new pixel value becomes slightly stronger while the rest becomes slightly weaker. If no Gaussian was close enough to the new pixel, a new Gaussian is added

using the new pixel value, and some set initial variance and weight. If the defined maximum number of Gaussians in the pixel model was exceeded, the Gaussian with the lowest weight is discarded. This ensures that old data is removed when it has not been observed for an extended period of time. [3] Improves upon this by using varying K, by continuously removing underrepresented Gaussians. The GMM method works well for constantly moving objects, and slow lighting changes. It can also model repetitive background motion like leaves in the wind fairly well, since a GMM captures multiple values. Figure 4 at the start of this section was made using a GMM based method. As you can see, rapidly moving foliage causes the method to fail. Furthermore, an object will be added to the background slowly if it stands still for an extended period of time, as will large moving objects with homogeneous colors (since a pixel will see the same color value even though it is a different part of the object). This means that a car stopping for a red light will become background within a few seconds. Fast lighting changes also cause a momentary distortion in the background subtraction quality.

ViBe [18] is a more recent approach. Instead of an abstract representation of pixel values like in GMMs, ViBe keeps a set of explicit sample values as history for each pixel, where samples can be values of the pixel itself or of the pixel's neighborhood. When deciding if a pixel is background or not, it checks if enough samples are close enough to the new value. If there is not a sufficient number of close samples, the pixel is considered foreground. The approach is conservative in its update mechanism, since it will never include a foreground pixel in its background model. If the pixel was a background pixel, it has a random chance to replace a random sample in the pixels history. In order to account for illumination changes in the background, or adding/removing a static object to the background, the pixel models in the neighborhood of background pixels are also randomly updated using the background pixel. The nondeterministic nature of this approach is fairly unique in background subtraction algorithms. ViBe's results suggest that it outperforms GMM-based methods. The use of neighboring pixels increases robustness against micro movement of the camera, which GMMs have a hard time dealing with. It also adds some robustness against the rapidly moving foliage problem.

The method proposed by Papazoglou and Ferrari [14] does not work on a microscopic (pixel) level like the previous two methods, but rather mostly on a macroscopic (full frame) level. The smallest entity used is a superpixel [20] (see Figure 5). Their method first uses the per-pixel method of optical flow to detect (partial) motion boundaries in each frame. Next, an inside/outside map is created using these boundaries. For each pixel, rays in eight directions with the pixel as origin are constructed, and the number of crossed boundaries is counted. An odd number of crosses indicates being inside an object. The map serves as an initial guess as to what pixels belong to moving objects. A GMM is created to serve as the estimated foreground model. A set of foreground models from previous frames is used to get a more reliable foreground model. The same process is repeated for background pixels and the background model. An energy minimization method is used to label a superpixel segmentation of the current frame, where the energy terms include object/background model, spatial and temporal terms. The method detects moving objects with impressively accurate boundaries, presumably due to the usage of superpixels. The complete algorithm is costly to run however. It seems the method has trouble with multi-object segmentation, as it has a single foreground and background model containing all objects. It likely also has trouble when new objects appear and leave continuously.

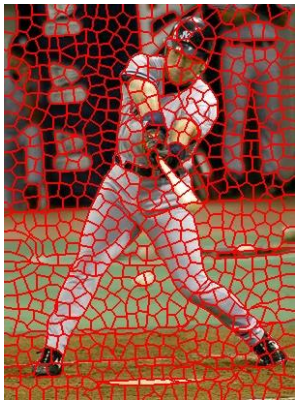

Figure 5. Superpixels.

2.1.2 No explicit background model

When no explicit background model is used for the entire frame, methods often utilize inter-frame differences to detect motion.

Optical flow [1] can be used to determine prominent moving foreground objects, assuming a static background. The optical flow between two consecutive frames consists of motion vectors per pixel, or per group of pixels. Figure 6 shows an example of how these vectors might look.

Figure 6. Optical flow.

These motion vectors indicate where the pixels moved to from one frame to the next. By thresholding the magnitude of the vectors, we can determine the prominent moving objects. There are a few issues with this approach if we want to accurately label every pixel as either background or moving foreground. First of all, there is no compensation for repetitive background motion since a static background is assumed, so moving leaves in the wind will continuously be detected as foreground even though they are uninteresting for most applications. Second, a large area with constant color will have no apparent flow, even though it may be part of a bigger moving object. This would require a separate system to fill in objects with 'stale' interiors, such as using optical flow to estimate the boundary of objects and then filling in the area contained by the estimated boundary. In conclusion, optical flow can give some initial indication of where movement is, but it has to be combined with more processing like in [14] to be robust for accurate moving object detection.

Meier and Ngan's method [5] is aimed at detecting video object planes (VOPs), for video encoding. Video encoding is the process of converting a video into a standardized digital format for playback with common video players. It often uses compression, and this is where the VOPs come in handy. [5] Uses Hausdorff distance minimization on edges of consecutives frames to detect moving edges.

Their algorithm works on edge images obtained by using the Canny edge detector on full video frames. The obtained moving objects in the form of edges are filled naively from left to right and top to bottom to obtain the final pixel mask for the object. This approach works moderately well, but it fills in the object between its edges in a simplistic fashion, potentially causing a lot of overestimation for concave objects. This is acceptable for VOPs since overestimation does not affect visual quality when encoding video, but we are looking for better accuracy since overestimation can cause visible artifacts when a recorded object is played back over another recorded object. It also relies on clear edges for proper detection, which may not always be available. Occlusion handling is also absent in this method.

Ensemble Video Object Cut [6] uses information from the entire video to do the segmentation. The method works by solving a large graph optimization problem. First, the Histogram of Oriented Gradient (HOG) features for image patches in each frame are calculated, with the same location for patches in each frame. These features are then clustered create the visual words, each of which then describes a HOG feature that is observed multiple times throughout the video. Using this bag of visual words, a histogram for each image patch in each frame is created. This histogram is used to compare patches in the graph-cut step. An 8-connected undirected graph between adjacent image patches is constructed, and each node is also linked to a global foreground and a global background node. Energy-minimizing graph cut is applied to assign either a foreground or background label to each patch. The energy terms incorporate the dissimilarity of the histogram of each patch, as well as some texture and shape information. The assumption is that foreground patches are similar to each other, and the same holds for background patches. The distance measure used between nodes includes the difference between the histograms of the patches and information of texture and shape. These measures are applied both spatially (neighboring patches in a single frame) and temporally (patches on the same location but in different frames). After the initial graph cut for each frame, the results of consecutive frames are fused to ensure more temporal consistency, and to improve accuracy overall. This method provides impressive results, even in highly cluttered and busy scenes. However, the biggest drawback is computation time, which is high due to the graph cut. It is also unclear how well the algorithm performs with many different objects continuously entering and leaving the scene.

2.2 Interesting object detection and tracking

The result of moving object detectors such as the ones described above can be used as a base for tracking objects. Individual objects can be recognized from the foreground mask, and these objects can be tracked. Usage of an explicit moving foreground object detection method before identifying and tracking individual objects is not necessary to track objects however, as there are plenty of methods that don't require such a prerequisite algorithm.

2.2.1 Using explicit movement detectors

[10] Applies background subtraction and connected component analysis to find objects. It then applies mean shift to track identified objects, and calibrates mean shift with the background subtraction when certain events occur. The background subtraction method used is a GMM-based one. A shadow removal technique is employed since they are undesirable in their application. 8-Connected component analysis is used to identify the individual objects in each frame. If the amount of components is not equal to the amount of tracked objects (which is zero initially), if the bounding box of two trackers intersects a single component or if a certain amount of frames (25 in the paper)

have passed since the last re-initialization, the trackers for the objects are re-initialized by matching them to the connected components. A non-occluded tracker matches a connected component if the distance between the two centers of mass is close enough, and the size of the tracker and component are close enough. If a connected component is not matched with any non-occluded tracker, it tries to find a match with occluded trackers using color histogram matching. A new tracker is initialized for the connected components that were not matched. To increase computational performance and to prevent the difficult problem of tracker-to-connected component matching, objects are tracked using mean-shift when no re-initialization step is required. Mean shift is an iterative tracking method, which searches the area around the last known location of the object. It tries to find the part of the image where the color distribution matches the object model the best. More detail on mean shift can be found in section 2.2.2. The result of the GMM foreground detection is used as a mask for the mean-shift tracker, which reduces the search space and keeps the tracker contained to moving objects where possible. There are a few issues with the approach. It requires stable and clean foreground detection, because of the way the method handles occlusions. Any noise introduced can throw the system off-track, and can also cause many unnecessary re-initialization steps. It appears to be useful only in controlled indoor environments.

Like [10], [21] also utilizes a version of the GMM moving object detection. The approach also uses connected component analysis to detect individual objects. The initial tracking is done by using a weak nearest neighbour matching between connected components and Kalman-filtered predictions of trackers. Unmatched connected components are assigned a new tracker. The bulk of the method comes after the initial tracks have been established, because the tracks are post-processed. The main goal is to link tracks that belong to the same object, accounting for occlusions and mergers. They use a number of link-probabilities that include temporal, kinematic (velocity), appearance and merge/split probabilities. The Hungarian method (a combinatorial optimization algorithm) is used to optimize the links. The results are fair, but will likely suffer greatly from noise since the initial online tracking is not very robust.

In [11], Kim et al. use their own explicit moving foreground object detector. Blob detection is used to detect individual objects or groups of objects. Blob detection differs slightly from connected component analysis since blobs don't require a completely connected set of pixels but rather allow some space between pixels. Blobs are then tracked by simply taking the Euclidean distance between the predicted next position of each blob (using the velocity) and the positions of the current blobs, and assigning each blob to the closest object. A similar approach is used in [12]. Tracking blobs is a little bit tricky however. Since the pixels in a blob are not necessarily connected to each other, there is a risk of grouping multiple interesting objects together as one "object". This approach does not work at all for reliable individual-object tracking, and has better use in larger scale group tracking. Figure 7 shows an example where blob tracking tracks multiple objects at once.

Figure 7. Blob-based tracking can track multiple objects as one (image taken from [11]).

Many algorithms that do not only track, but also accurately segment moving objects, work in a semi-supervised manner. This means that a user is involved during the tracking and segmenting process, to steer the algorithm (but the user is not completely segmenting and labeling each frame manually) as opposed to doing it fully automatically. Video SnapCut [15] is an impressive but supervised video object segmentation algorithm. Object initialization is done by having the user provide a relatively accurate mask for the object they would like to track. This mask can be quickly generated by using some auto-selection tools, such as the Quick Select tool in Photoshop. The algorithm then generates a set of small, overlapping windows along the object boundary. The windows each have color, confidence and shape values to classify foreground/background probabilities in their local small section of the boundary. These windows are then predicted in the next frame using motion estimation (global transformation of SIFT features and local optical flow). The actual object boundary within the predicted frames is refined. This is done by using the motion estimation to warp the old boundary, which serves as a decent initial guess. Then the best option between the old color model (including the history so far) and the new color model induced by the predicted boundary is used to refine the boundary. Iteration is applied to further refine the mask within a single frame. To further improve accuracy, multiple keyframes can be segmented by the user, and forward/backward propagation is used to merge path from different keyframes. Finally, a user can post-process the frames to fix errors in the object boundaries. While the method yields good results with accurate object boundaries, it still requires a user to fix minor mistakes to prevent drift, as well as a user for initial segmentation.

2.2.2 Without explicit movement detectors (tracking by detection)

One of the more known methods is Mean-Shift [7]. Mean-Shift tracks an object model given its position in the last frame. It assigns a weight to every pixel in the old object window by comparing their values to the object model (color histogram). Once all weights have been calculated, a weighted average of the pixel positions (the mean) is calculated, and the object region is moved to be centered at that mean. This process iterates, until the new position is close enough to the position of the previous iteration. Usually, pixels further from the object's center are weighted lower, since they are more likely to be part of the background. The object model has to be re-evaluated every so often, to account for changes to the object's color histogram. To account for size changes, both a smaller and larger object region can be compared to the regular final region after iterating. The best of the three results can then be picked, and the size of that result is used in the next frame. Figure 8 shows an example of mean-shift tracking.

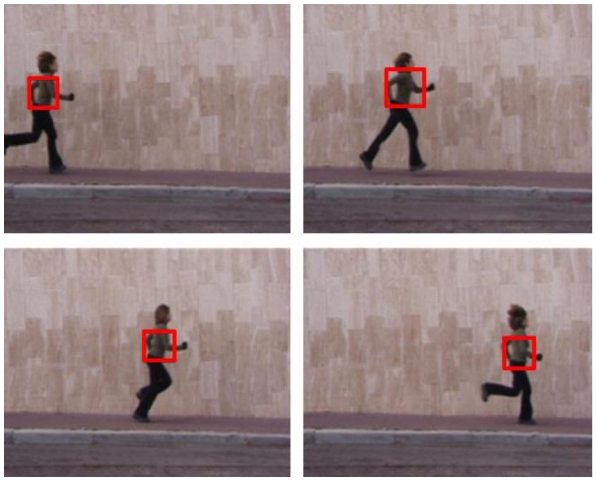

Figure 8. Mean shift tracking. Note how the tracker is not tracking the exact same part of the body each frame.

An issue with mean-shift is that there is no system in place for occlusion handling. Trackers can also easily start tracking a similar object or a similar piece of background. This can be especially true in our application, where mean-shift could easily get stuck on a parked car with similar colors as a moving object. Drift is also an issue, because when the object model is updated with slightly wrong information, this slight inaccuracy can propagate throughout the video, causing the tracker to permanently lose the object. Mean-shift also does not accurately track object pixels, but rather the general area the object resides in, namely the search window. This can be any pre-defined shape such as a bounding box or ellipsoid, but such a shape often does not accurately describe an object (as seen in Figure 8, where the tracker tracks slightly different parts of the body each frame). This means that a secondary system would have to be used to actually segment the object's pixels from the area. Mean-shift does not have a system in place to detect objects in the first place, either. [8] Tries to alleviate some issues with mean-shift by adding particle filtering. For each new frame, Mean-Shift is applied, as well as a particle filtering method with a motion prediction model. The better of the two results is used (Ensemble approach). [9] Divides the object's bounding box into multiple sections and applies mean-shift to each section separately, which helps in dealing with objects that have independently moving parts, such as the limbs on humans.

[13] Does something quite different from most tracking algorithms, since it attempts to exploit spatial constraints between all the visible objects for their tracking. Objects are represented by HOG features. Springs are defined in a graph-like structure between objects. This graph can be a minimum spanning tree, or a graph where every node is only connected to a dummy node located at the center of all nodes (the center is at the average position of all the nodes). A local maximization technique for each new frame is used to find the new optimal joint configuration of all objects. This means that the location of each object influences the location of all other objects by some amount. This approach works best when tracking objects that have similar movement patterns or objects with dependent movement patterns, which is often the case when tracking cars. However, their approach is limited in that it uses the same size bounding box for a single object during its entire lifespan. This does not work well with large changes in scale. Like many of the other non-explicit foreground methods, it does not accurately track an object's pixels, but rather a rough area. It also does not appear to have a method of detecting objects in the first place.

[16] Also tracks using similar user supervision as in [15]. A user initially defines a rough stroke across the boundary of the object of interest. The algorithm then attempts to refine this rough boundary into a tight boundary using the edges of homogeneous regions in the image that overlap the stroke. The boundary is then tracked through time by motion estimation of the boundary and iterative refinement. During the tracking the user may still have to make adjustments to fix inaccurate boundaries, which could otherwise propagate further, possibly resulting in a complete loss of accurate tracking. This method is less powerful than [15], and still requires the same, if not more, supervision.

3. The proposed segmentation algorithm

In this section, the proposed method for tracking and replaying cars will be discussed. Section 3.1 will give an overview of the method, discussing the different steps at a high level along with their motivations. Section 3.2 and 3.3 will focus on the specifics of each step, and their associated settings and parameters. For a table containing all parameters, refer to Table 1 in section 4.

3.1 Overview of the method and motivation

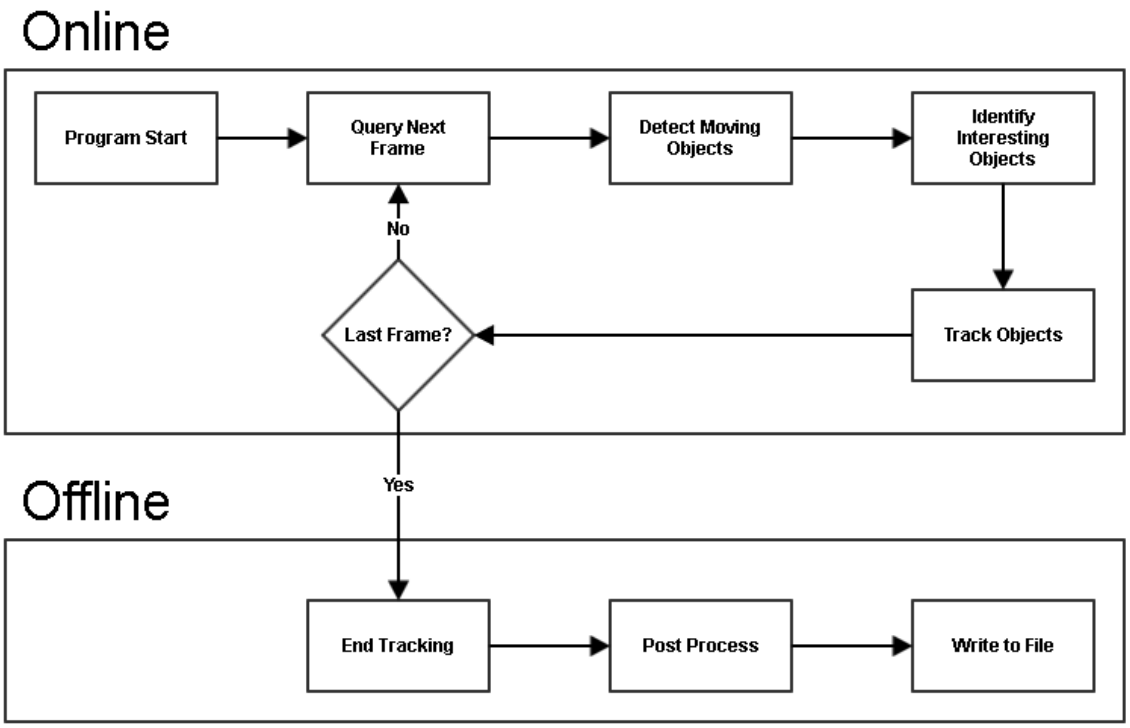

Figure 9. Overview of the method.

The proposed method (Figure 9) expands partially on Adaptive mean-shift for automated multi object tracking [10]. What we use from [10] are GMM-based moving foreground detection and object detection using connected components, and a small part of the tracking (but not the actual mean-shift portion). Some extra tracking steps were added, as well as a large amount of post-processing.

The tracking and segmentation methodology can be separated into two parts: “Online” tracking in each video frame and “offline” post processing of the tracks. By doing this, we only have to process

each video frame once to obtain our interesting data, namely frame data of interesting objects and their paths. This typically reduces computation time, since decoding our particular video files is rather intensive due to frame rate and resolution. The offline processing then only has to deal with the generated paths and (often tiny) images of objects, which is all that is required to enhance the tracked objects' paths.

3.1.1 Tracking in each video frame

The entire video is processed once. The frames of the video are handled in chronological order. A GMM-based foreground detector [3] is updated with each new frame, and we obtain the foreground mask from this detector. The mask is then refined by morphological filters and a flood-fill method to filter noise and fill in holes in objects. From the refined mask, 8-connected components are detected. Components that are too small or too spread out over their bounding box are filtered out, since they are likely noise. The connected components are then matched with tracked objects in the previous frame, and occluded objects. This matching is based on size, color model and distance. If no match can be found, a new tracked object is created for the component. When the last frame has been processed, post processing starts.

3.1.2 Post processing

The result of the per-frame tracking described above is a set of tracks, corresponding to different objects. However, the tracks are often far from perfect. Issues such as trackers jumping to a different object, trackers losing their object even though it is still there, and partial occlusions splitting objects into two separate objects are some examples of thing that can go wrong. In order to fix some of these issues, several post processing steps are applied. Tracks are split into two separate tracks if the direction changes abruptly (tracker jumped to a different object). Inter-car occlusions are detected and fixed, to prevent two cars in one detection. This double car detection can occur when two cars overlap in the video frame, causing them to temporarily form a single connected component and hence be detected as one car during the online phase. Two trackers that belong to the same object, caused by, for instance, an object being split by a tree, are linked together. Objects that are moving from/towards start/endpoints are extrapolated towards those points if the object disappears too soon. Finally, when the tracks are final, any visual data in occluded frames is approximated by linear interpolation between the two closest available frames. Figure 17 in section 3.3.1 schematically shows the automatic post processing steps.

3.1.3 Motivation

The reason why we use connected component-based tracking using a foreground mask is that cars tend to be stable in this respect. By this we mean that in the foreground mask, the cars will likely be visible just as well as they were in the last frame, preventing total loss of tracking or major errors in accuracy of the car pixels. As a result, the connected component of a car also remains similar throughout frames. The post processing steps are added because there are several errors that can't be solved the first time a frame is processed, as described in section 3.1.2, but that can be solved once we have more data.

3.2 Tracking in each video frame

This section will describe the frame-by frame processing in more detail.

3.2.1 Preparing each queried video frame: Equirectangular video

The current system uses equirectangular panoramic videos. This means that the left border of the video connects perfectly to the right border, so things "wrap around" from left to right or vice versa. In order to make tracking objects that exhibit this wrapping easier to implement, each frame is expanded on both sides. This is done by taking a piece of the left of the frame, and pasting it to the right, and vice versa (see Figure 10). In all of our testing, these pieces were set to 20% of the width of the original frame. This should cover all but the largest close-range trucks. When an object is larger than the extra section, it is possible that the object is split into two objects with the current setup. In scenarios with large screen-covering objects, one may consider increasing the size of the extra section. In all following sections it is assumed that operations work on the expanded frame, unless explicitly stated otherwise.

Figure 10. Equirectangular frame. The green border indicates the original frame, the red border indicates the pasted wrapping portions. The blue box indicates a car that is now visible twice.

3.2.2 Optional initial user input

Before tracking starts, the user can give some guides to help accuracy and computational efficiency. These guides are optional.

First of all, the user can paint a rough mask over the area in which desired objects will appear. This mask is used to filter out unwanted foreground pixels returned from the foreground detection. The mask reduces computational load significantly, because the expensive connected components filter later on will have fewer pixels to consider. The mask also reduces the amount of false positives, since objects outside the area of interest will not be considered during the next processing steps. This mask is painted on the non-expanded frame, and gets the same expansion treatment as the video frames before tracking starts. This ensures that the expanded pieces on both sides are consistent with their copies in the original mask, since a user likely would not paint the expanded sections exactly the same as the original.

The user can also point out spawn points for tracked objects. By spawn points we mean points where a car can appear or disappear in the video. By appear we means that the car has not been visible in the scene before that point, and by disappear we mean that the car will never be visible again after that point. Examples of spawn points include large buildings (but not all buildings are necessarily technical spawn points) and distant points on the horizon (see Figure 11 for an example).

These spawn points will later be used to fill in the final frames of cars that get too small for detection right before they disappear completely at a spawn point. The spawn points are also placed on the non-expanded image rather than the expanded one.

Figure 11. Example of a spawn point.

These user inputs can be given by even the most inexperienced of users. Painting the mask and assigning spawn points can take less than a minute, but the input still gives some valuable information to the algorithm.

3.2.3 Moving object detection

Moving object detection is done by the widely used Improved Gaussian Mixture Models method (GMMs)[3]. This technique only works with static cameras. It is robust against slow illumination changes, but it does not cope well with sudden illumination changes or sudden changes in scenery. The method does adapt to these kinds of rapid changes, but at a slow rate (which can be adjusted). Another problem is that it needs a few seconds of video at the start to learn the background, but this is not a large issue for our purposes, since any cars that are already present at that time can't be detected properly anyway since too much information is missing. The used implementation of GMMs also has the option to detect shadows separately to filter them out (following the original paper), but we actually do want shadows in our detection since we need to replicate them when we replay the cars.

The result of GMM foreground detection is a binary foreground/background mask (Figure 12.A). Any pixels not in the user-defined mask are discarded (Figure 12.B). The mask is often noisy due to camera noise and small background movement (i.e. foliage). The morphological filters erosion and dilation are used to filter out the vast majority of the noise. More specifically, opening (erosion and then dilation) and closing (dilation and the erosion) are performed (Figure 12.C and 12.D). The order of opening/closing depends on the scene. For scenes with mostly distant cars, closing is done first in order to not erase cars because of their small size. Otherwise, opening is done first. The amount of iterations of the operators is scene dependent, where scenes with larger cars can benefit from more iterations (but too many iterations can cause severe overestimation). The application of the morphological filters also has a negative side effect, as it makes the mask more blocky (because square kernels are used). This is not a big problem in our application though, as a slight overestimation is preferred over missing pixels in vehicles. Only chaotically changing background objects such as fast swaying trees might unintentionally pass the filter, but they likely won't produce tracks because of their size/duration (filtered out later). The foreground mask is further refined by

using a flood fill technique to fill in holes in objects (Figure 12.E). This is done by first creating a flood fill mask using a background pixel (the pixel at (0,0) in our tests) as a source point. Then, the inverse of this fill mask is the new foreground mask, which will have fully enclosed holes in objects filled. This is fine, since we don't expect there to be holes in cars. Donut shaped-cars will have the hole inside of them overestimated as being part of the car using this method, however.

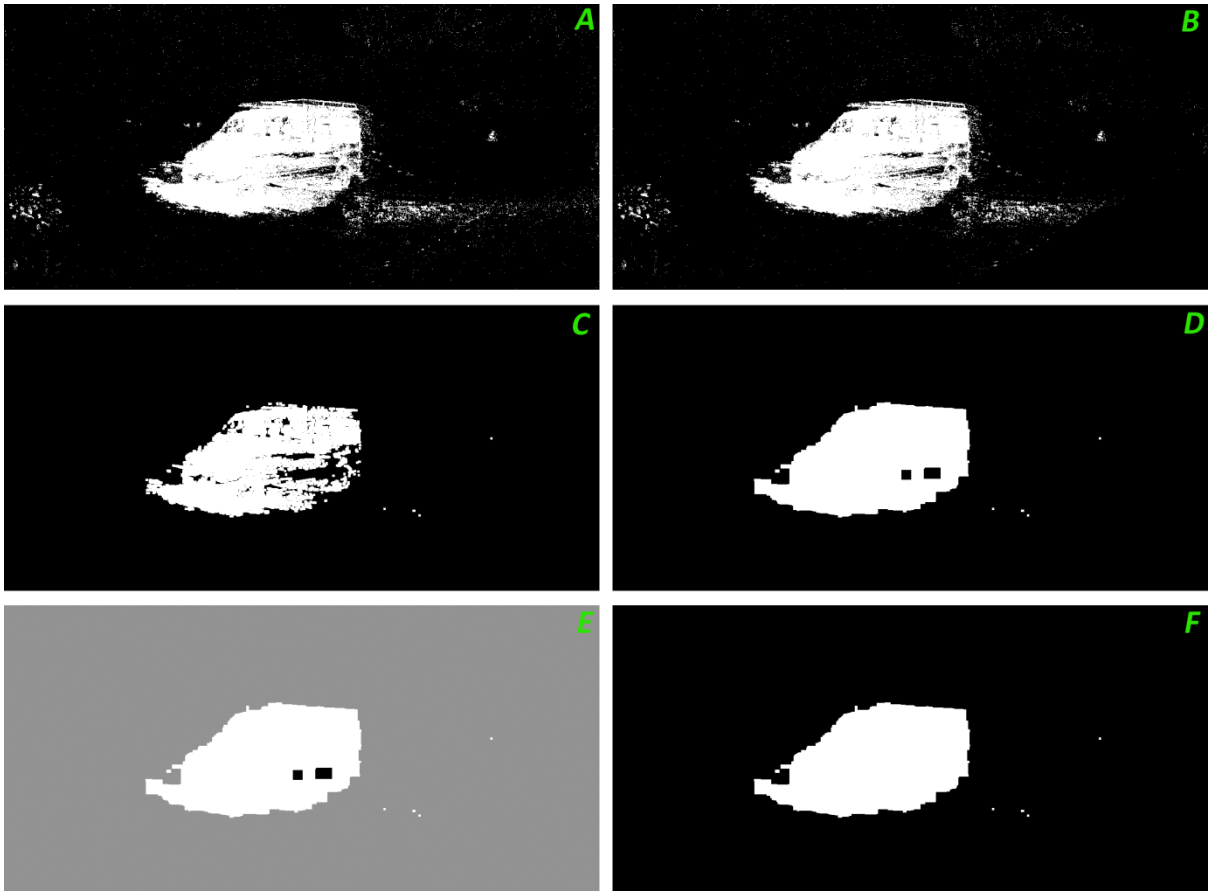

Figure 12. Foreground detection refinement. A) Raw output from GMM; B) User mask application; C) Opening; D) Closing; E) Flood fill (grey); F) Final result.

3.2.4 Interesting object identification

8-Connected component analysis on the foreground mask is used to detect where interesting objects are. An 8-connected component is a set of pixels where you can follow a path from one pixel in the set to any other pixel in the set by only stepping to adjacent pixels in 8 directions, and only passing through pixels in the set. Since cars are solid, connected components should be a good representation. With good enough foreground detection and refinement, the connected component of an object will be mostly stable throughout the video. By stable we mean that the connected component of a single object will represent the object with the same accuracy, which means that for instance visible parts of the object are not suddenly gone from its connected component while they were present in the previous frame. Another way of saying it is that the connected component of a car does not change drastically in consecutive frames. Blob detection would be an alternative if cars could consist of multiple connected components, but this is often not necessary and can result in misclassification of multiple objects as a single object.

Once connected components are detected, we discard the ones that are too small (area smaller than some threshold), since they are most likely small bits of noise that have slipped past the background

refinement procedure. We also discard the ones with a too small (bounding box area) / (actual pixel area) ratio. This ratio determines how "filled" the bounding box is (Figure 13). For cars, which are fairly rectangular and solid objects, we can expect this ratio to be fairly high. Connected components along the edges of the background caused by tiny camera motion however, will have a lower value and will be filtered out.

Since we are working with the expanded frame, connected components can be detected twice if they are in the copied parts of the frame. Since the copies will always be exactly the width of the original frame apart, we can detect such copies and only work with one. In our case, the left one is always picked for consistency, but either left or right would work fine as long as one is discarded.

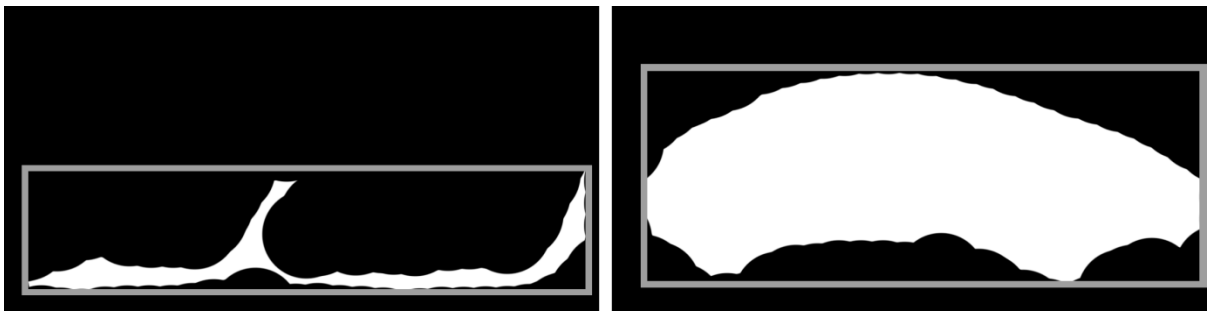

Figure 13. Invalid connected component due to low pixel/bounding box ratio (left), valid connected component (right).

3.2.5 Object Tracking

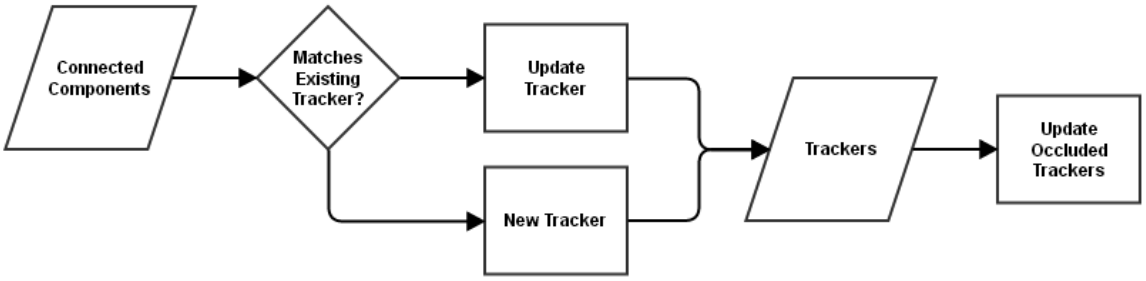

Figure 14. Overview of object tracking.

A tracker for an object holds the known information of the object during tracking. For the path, the bounding box and center of mass for each frame are stored. For the visuals, the subsection of the video frame containing the object is stored, along with the foreground mask in that section. A running weighted average of the color histogram (with bins in all channels of the YcbCr space) over the past few frames is also kept. When a tracker is created, it also stores its starting frame number. Occlusions intervals in the path are stored, as well as whether the object is occluded right now or not. If the object is currently occluded, the last known position, velocity and acceleration are stored to allow for prediction during the occlusion period.

In order to track objects, we first try to match the connected components in the current frame with the currently active object trackers (if any exist already, initially there are no trackers). Each connected component is compared to every active tracker. From all the matches per connected component, we pick the best matching tracker to describe the component. Priority is given to

trackers that are not currently in an occluded state. Objects are matched based on size of the bounding box, position and color histogram (Figure 15) as described in more detail below.

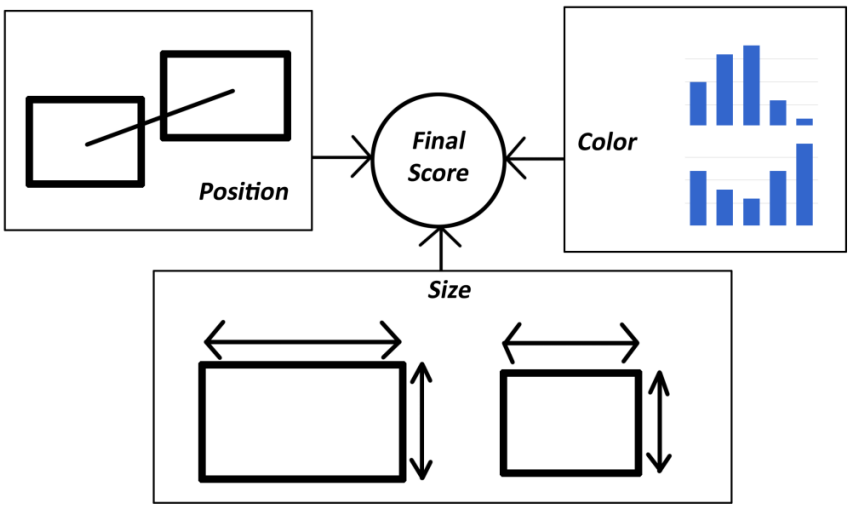

Figure 15. Distance score.

For the size score, we want to compare the sizes of the two objects and threshold the ratio between them. This allows us to filter out matches where the connected component is not of the same size as the car we are tracking, and therefore is not likely the same car. For small objects a different threshold is used than for larger objects, instead of using a single constant threshold. The reason for this is that smaller objects are influenced more by noise than large objects, which can result in larger relative size changes. By using a separate size change value for smaller-sized objects, we can partially compensate for that.

The size score is calculated as follows. Refer to Figure 16 for a visual representation. First of all there are two size-ratio thresholds: a large-size threshold and a small-size threshold. Both thresholds have a threshold value and an associated size-change value. When calculating the size score between two objects, we compare the bounding box size of the larger of the two objects with the threshold sizes to find the size-change value that we will use in this particular comparison. If the size of the small object was smaller than the small threshold size or larger than the large threshold size, we clamp the size-change value back to the closer threshold. Otherwise, we linearly interpolate the size-change value between the large and small thresholds. The size-change value defines the maximum size ratio between the smaller and the larger of the two objects. We then compare the size ratio (larger object size / smaller object size) to the custom maximum size-change value we calculated. If the ratio is larger than the size-change value, the objects are not considered similar (and any further similarity scores are not calculated). If the ratio is smaller, the size portion of the similarity score is finally calculated as (1 – size ratio / size threshold), giving us a normalized similarity score between 0 and 1

for any pair of objects.

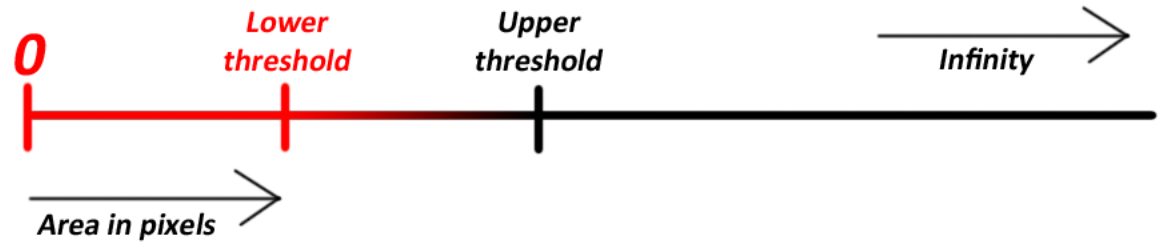

Figure 16. Size threshold interpolation.

The positional distance score is determined using the maximum per-frame distance value in the parameter set. This allows us to filter out matches where the connected component is so far away that the car we are tracking could not have moved that far in one frame. We use the center of each object to calculate Euclidean distances. If the object has connected component information for the frame where we want to calculate the distance, we use the center of mass of the connected component. In the case of occlusion, no connected component information is available. In this case, we use the center of the predicted bounding box instead. If the centers of the two objects are further away than the given maximum distance, the objects are not considered equal and no further scores are considered. If they are closer, then the similarity value is determined as (1 – distance / maximum distance), giving us another normalized measure between 0 and 1.

Color similarity allows us to filter out the connected components which have a significantly different color distribution than the car we are tracking, since we do not expect wild color changes in consecutive frames. To calculate color similarity, the Bhattacharyya coefficients between the two objects' color histograms is used, in the same fashion as [10]:

$$S(p,q) = \sum_{i=0}^{b} \sqrt{p[i] * q[i]}$$

Equation 1. Bhattacharyya coefficient.

Where *S(p,q)* is the similarity between two histograms *p* and *q*. Each histogram is made up of *b* bins, and *p[i]* is the value of bin *i* in histogram *p*. Only foreground pixels are used for histogram calculation. The histograms use the YCbCr color space and all three channels are considered, following [10]. They are calculated using a kernel with an Epanechnikov profile as described in [7]. This kernel gives lower weights to pixels further from the center of the kernel. This often helps with histogram accuracy since edge pixels of objects are more likely to be misclassified background pixels, pixels with video compression artifacts or noisy pixels. The Bhattacharyya coefficient gives us yet another normalized measure between 0 and 1. Like with the other two scores, the color score can also veto the similarity between two objects if they are too dissimilar.

Each of the three scores also incorporates a slack modifier when we are looking at an occluded tracker. Since prediction is not 100% accurate and prediction does not update the histogram at all, we are more lenient when matching a connected component with an occluded tracker. The size and distance share a slack value, which multiplies the critical thresholds with some factor. This reduces the change that misprediction causes a rejection of a real match. The histogram distance threshold

has a separate factor. This makes cumulative color changes throughout occluded frames less likely to cause rejection of a match. Like most parameters, these thresholds have been determined by an initial educated guess followed by some empirical refinement. However, they have not yet been explored to a large extent.

Finally, all three scores are added, with equal weights, resulting in a final similarity score for the pair of objects (see Equation 2). No experimentation has been done yet with different weights for the scores. All three of the scores are normalized between 0 and 1.

$$score_{final} = score_{size} + score_{position} + score_{color}$$

Equation 2. Final similarity score.

If there are multiple valid matches, the match with the highest score is chosen since that is most likely the right one. The tracker is updated with the information of the matched connected component. We modify the color model of the tracker with the new information, by linearly blending each histogram bin using weights for the old and new model. The new histogram is then normalized to prevent propagation of rounding errors, by simply summing up all bins and dividing each bin by the sum. Updating the histogram in this fashion means the tracker can adapt to changes in the color model of the object, which is something that happens almost guaranteed for objects at close to medium range as we will see different sides of the object as it passes by. Rapidly changing reflections on cars also change the histogram significantly. The center of mass of the connected component is used as the new position for the tracker since that is more accurate than the center of the bounding box. The frame data in the bounding box of the connected component is used as the new frame for the tracker. This frame data also has the foreground mask applied, so that we only store the foreground data of the object. The foreground mask is blurred using a given intensity, so that the edges of the car frame that we store are slightly smooth instead of blocky.

If no tracker was matched with the connected component, we create a new tracker for it. Any trackers that did not get a matching connected component are marked as occluded. Occluded tracker will get predicted every frame.

Predicting the new position of an occluded object goes as follows. When an object first becomes occluded, the frame number at which this happens is recorded to later fill in the occluded intervals with estimated data. If an object becomes un-occluded, the occluded interval is closed by recording the frame number where the object appeared again. When an occlusion occurs, the velocity direction, speed and acceleration are also recorded. These are calculated by fitting a line to a given length of the object's path using least-squares, and calculating the average speed and velocity on the line. This smooths out jerky positions due to noise. Positions in the path that jump to the other side due to the frame wrap-around are compensated for when fitting the line. In each occluded frame, these values are used to predict at what position the object should be assuming none of the values have changed. The size of the bounding box and the object model histogram remain unchanged during prediction.

When a certain amount of occluded frames have passed, the tracker is considered gone from the scene, and will no longer be considered during the on-line tracking. Any pending occlusion intervals are discarded since the car should have disappeared, so there is no data to estimate.

3.3 Post processing

Once the entire video has been processed frame-by-frame, post processing begins. The majority of the post processing is done automatically, while a small part has to be done manually by the user.

3.3.1 Automatic post processing

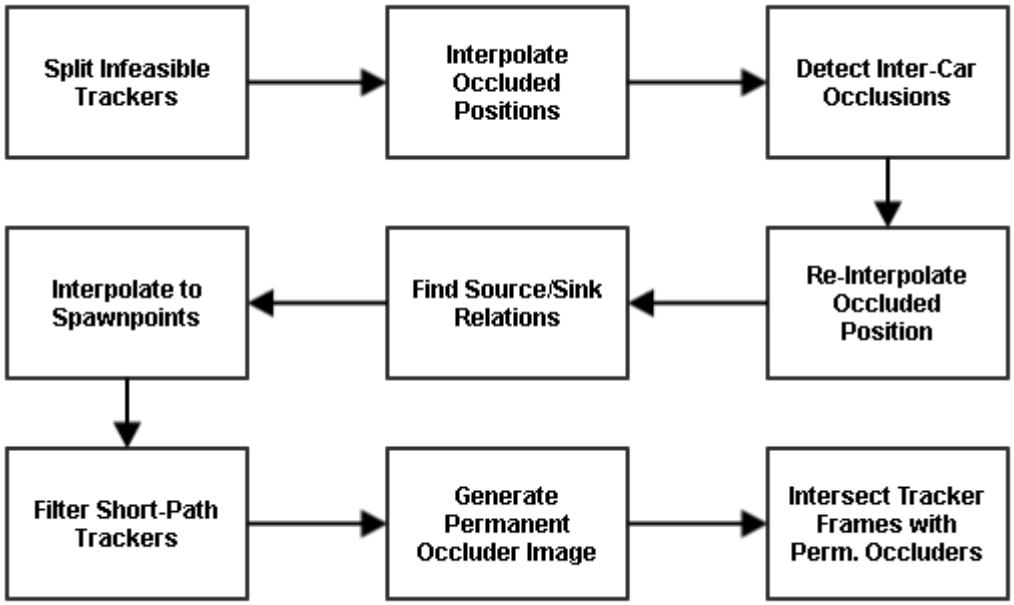

Figure 17. Overview of automatic post processing.

Before the data is processed, any lingering processes with respect to occluded trackers are finalized. This means removing all consecutive occluded frames that end in the current frame, as we will not continue tracking to find the end of the occlusion event. This means that there will not be information at the end of the occlusion interval to interpolate with, so the frames are not usable. After that, the actual post processing can start. Noteworthy is that the order in which cars are processed can in some cases affect the outcome.

First, every tracker's path is traced, and trackers that suddenly start moving in the opposite direction are split into two separate trackers at the direction-change point. This is done because we likely have a case we like to call the relay effect. This effect can mainly occur when two similar cars' connected components merge, causing a tracker to switch targets and track the other car instead. A parameter can be specified at which angle the tracker is considered "moving in the opposite direction", as well as a parameter to determine how long the car has to move in the opposite direction in order to consider the initial change a split point. We don't want to create a split point at every frame where an opposite direction is detected, because noise in the background subtraction can easily cause a single opposite-direction movement even though after that movement stabilizes again. The exact method of finding the splits therefore processes every point in the path in order, and remembers where a potential split occurs between two consecutive frames. After such a

potential split, it looks ahead for a certain number of frames to determine that the next frames also move in the same “wrong” direction. If such a set of consecutive wrong frames is found, a split is created.

Next, we interpolate the bounding boxes in all occluded intervals of all cars, discarding the predicted path points that were already there (Figure 18). Both position and size are interpolated by linearly interpolating each corner of the box between the two reference frames. This process is done to obtain smooth paths. Without this interpolation, it is often the case that the prediction drifted a bit, and when the tracker re-appears, a noticeable jump in position can occur.

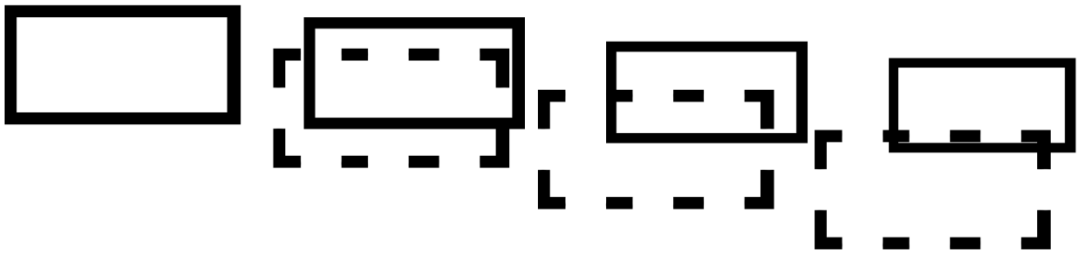

Figure 18. Incorrect prediction frames (dashed boxes), correct interpolation afterwards (non-dashed boxes).

Using the resulting complete paths, we detect inter-car occlusions. This is done by checking for bounding box overlap between all car paths that overlap in time. New occlusion events (the same type that gets created during online tracking when an occlusion occurs) are made using these occlusions, and the new occlusion events are then re-interpolated as described in the paragraph above. This step attempts to prevent the situation where one car is behind another in a single connected component, yet the tracker for the front car still tracks the connected component in question since the front car dominates the color histogram. This situation would result in a second car appearing briefly in frame of the front car if we would not fix it. Not all instances of this scenario will be prevented, since it does require accurate and complete tracking of both cars before and after the overlapping frames.

The next step is to find what we call source/sink relations. What can happen during tracking is that a car is split in two by a pole or tree (Figure 19). This can result in two separate tracked connected components that describe the same car. Since the car will first approach the pole, one of the two components will spawn while the other component of the car is still active and tracked. A new tracker will be created for the new component that appears on the other side of the pole. We call the newly spawned component the sink of the older component, while the older component is called the source. We try to detect such relationships, because without such detection, a single pole could render a scene worthless since every car could get chopped up in two unlinked trackers by it. The detection is done by comparing each tracker to every other tracker. We consider the cases where the starting frame number of one tracker coincides with a frame number where the other tracker was active. If such a correspondence is found, a source/sink relation could occur. We calculate the velocity direction for both cars at the considered frame, and check if the angle between the two velocities (calculated as described at the end of section 3.2.5) is below a defined threshold. We also calculate if the distance between the two bounding boxes is below a certain threshold (note that we use bounding box distance instead of distance between centers). This distance threshold

defines the width of poles/trees that the system can deal with. We also compare the histograms of the largest (which is assumingly the most accurate) frame in both cars' paths, to see if the colors are similar using a separate threshold. We also check if the sink does not "spawn" behind the source (otherwise we would create source/sink relations at spawn points, where cars spawn behind each other). If both velocity, distance and color thresholds are passed, we create the source/sink relation. This means that whenever the source is being played back, an instance of the sink will be spawn at the right frame.

Figure 19. Source/sink situation caused by a pole.

As a last change to the paths, we try to interpolate from or towards user defined spawn points (if any have been defined). For every car's first and last frame, we check their velocity and acceleration at that point (calculated as described at the end of section 3.2.5). The first frame's velocity and acceleration are inverted, and we check if the cars was coming from a spawn point. The same is done for the final frame, but we check without inverted velocity/acceleration if a car was moving towards a spawn point. If we find a matching spawn point, we extrapolate the path towards the spawn point. This is done using the calculated velocity and acceleration, and moving the bounding box into the direction of the spawn point with said velocity and acceleration. The size of the bounding box remains unchanged, as we will use the last known frame for these included frames. If multiple spawn points match, the closest one is used.

Now that the paths are finalized, we discard trackers that are too short. This is done by considering every path from each sourceless tracker, and recursively checking each path through a series of sinks (if the tracker has any sinks). This basically means we check strings of source/sinks. If a string is long enough, all trackers in the string are flagged as valid. If all strings have been checked, any unflagged trackers are discarded. This filters noise-trackers since they generally only exist for a fraction of a second. It also filters out inaccurate/highly fragmented car detections, that just define a small fraction of a car's path and have not been linked to the "main" tracker of the car. The noisy frames will be interpolated since the main tracker will lack information there, and therefore have an occlusion interval for the frames.

Finally, we build a permanent occluder image (Figure 20). This image defines the pixels that were never classified as a real car, given the final paths after invalid cars have been discarded. A blank image is created with the size of the video. We then take all binary non-occluded frame masks of every tracker, and use the OR operation between its pixels and the corresponding pixels in the occluder image. The result is a binary image that shows where a car has been definitely detected. We don't use the refined foreground detection result for these frames however. We use the raw mask from the GMM foreground detector. The refined mask is just for more stable tracking and noise removal, but it also influences the permanent background mask which is undesirable.

Figure 20. Partial permanent occluder image. Notice the pole on the left.

3.3.2 Manual post processing

The manual part of the post processing is done by having the user replay every detected car. The user can then determine if the car is good enough to keep or should be discarded. If the user deems a car as being good enough (accurate enough detection) for his specific application, he writes down the spot in the Z-ordering of the car. This Z-ordering basically has an integer number for each lane, with the furthest lane being 0 and every other lane being 1 index higher than the previous furthest lane. This Z-ordering is necessary when replaying cars, to prevent inconsistencies by drawing cars over other cars that should be further away. While this is not a fool-proof method since cars can change lanes, it is sufficient for our purposes since any conflict can be circumvented by making a conflicting car spawn slightly earlier or later.

3.4 Creating new videos

Now that the segmentation is done, we can prepare the cars for inpainting and then we can create our custom videos. First, any missing or unusable car frames are fixed. Next, a background video is created. Finally, a user creates a scenario by determining when each car should spawn, and he manually checks if there are no collisions.

3.4.1 Filling in missing or unusable car frames

When the segmented paths have been processed by the automatic processing described in section 3.3.1, we still have to generate the missing frames of during regular occlusion intervals and inter-car occlusions. Any occluded frames are linearly interpolated between their nearest known frames. The result of this can be seen in Figure 21. The longer the occlusion interval, the worse the interpolation since the two frames that are used as source data will be more different from each other. We also apply the AND operator to the interpolated frames' alpha channel and the binary permanent occluder image built at the end of section 3.3.1. This prevents interpolated frames to be drawn on top of the permanent occluders (such as trees and poles in front of the cars) . It does not work for

occluders that only occlude some cars, but not others (such as poles between two lanes). For spawn point interpolation frames, we slowly fade from 0 alpha for start frames, or to 0 alpha for end frames. This makes it so the car does not just pop in or out of existence when it reaches a spawn point.

Figure 21. Inpainting. A) The frame right before an occlusion interval. B) An interpolated frame during the interval (interpolated using A and C). C) The frame right after the occlusion interval.

3.4.2 Creating a background video

Next we need a background video, which is a video where no cars are present. The video is processed frame by frame once more, this time with just the background subtraction active. The parameters for the background subtraction and refinement can be set to over-estimate more than during tracking, since our main goal now is to remove all moving cars from the video completely. Major overestimation is significantly less likely to cause issues here. The optional mask described in section 3.2.2 can also be applied here, to ensure that for instance slow moving pedestrians are not causing artifacts in the background video. Using the background subtraction, we use the most up-to-date background information that we have as frames for the new video. When a new frame is processed, all pixels identified as background will update this up-to-date background with their new value. To make sure that we don't have black holes in the background frame at the start (when some pixels have never been background yet), we initialize the background with the first frame. Every time a frame has been processed, the new background is used as a frame for the background video.

Since there may be visible cars in the first frame of the video, these can show up in the background video. It is therefore necessary for a user to look at the background video, and discard the initial part of the video that still contains these cars. It can happen that somewhere during the video, some minor glitches appear when sudden lighting changes occur, or when a shadow of a car is not detected completely. A user can also choose to steer clear of these parts of the background video by selectively picking a part of the video without glitches.

3.4.3 Creating a scenario

Now that we have a background video and cars without missing frames, a user can create scenarios by defining when each car should spawn. He simply gives each instance of a car a timestamp, and the car will be inpainted starting from that point in time. Multiple instance of the same car can be active at the same time. Once the timestamps have been assigned, the user can render the full video, and check if there are no conflicts as described in section 3.3.2, and that there are no cars hitting each other from the back because one was driving faster when it was segmented. The video is then ready for use. In the specific case of the VCNS, the user also has to create a foreground mask for a single video frame. This mask is used to determine where the airplane is visible, so that the plane can disappear behind objects instead of being drawn on top of everything.

4. Results

To determine the quality of the tracking and replay solution, both a subjective user study and an objective study using manual car segmentations were executed as mentioned in the beginning of this section. For the user study, 11 videos were used (1 control and 10 test videos), to keep the time of each user test manageable. For the objective study, all 15 filmed videos were used. The videos were shot on a few different locations, with situations where cars were far away (20+ meters), medium distance (~10 meters) and close by (<5 meters). A few videos were explicitly filmed with poles or trees in the way to test the source/sink mechanism. The studies will be discussed in section 4.2 and 4.3 respectively.

Each of the 15 videos used in these tests were shot using 6 GoPro Hero4 Black cameras. The cameras are held together by a 3D-printed rig. Autopano Giga was used to create a panorama template for the camera rig for each video, and Autopano Video Pro was used to stitch the 6 videos into a single equirectangular panoramic 360 video. The individual GoPro videos were shot with a resolution of 1920x1440 at 80 fps, but due to limitations of the stitching software, the panoramic video was stitched with a resolution of 3840x1920 resolution at only 60 fps.

Every video has been processed by the proposed method once. Two sets of parameters were used for processing: One set for videos where cars were expected to be relatively far away (in our case about 35+ meters) and therefore be relatively small in the video frame, and one set for situations where cars would primarily pass at short distances (a few meters) and therefore be relatively big. Table 1 shows the used parameters. Many parameters are shared between sets since they are not thought to be significantly affected by car size. The parameters were empirically tested during development, but are by no means perfect. A detailed discussion on the problems with the parameters can be found in section 6.

Parameter	**Shared**	**Far away setting**	**Close by setting**
Expanded frame wrap-around %	20		
Foreground detection [3]			
History length	350		
Threshold	8		
Learning rate	disabled		
Foreground refinement			
Closing iterations		1	4
Opening iterations	1		
Closing/opening first		closing	opening
Floodfill origin	(0, 0)		
Blur iterations		2	8
Online tracking			
Minimum connected component area (pixels)		100	400
Maximum connected component (pixels)	2000000		
Minimum connected component density	0.3		
Bins per color channel	32		
Closeness similarity threshold		50	200

Size comparison thresholds (see 3.5.2) [area; threshold]	Large [1000; 3] Small [10000; 1.5]		
Histogram similarity threshold		0.9	0.85
New histogram blending weight	0.3		
Maximum occluded time (seconds)	1.2		
Occluded distance threshold slack	2.2		
Occluded histogram threshold slack	0.85		
Path length for velocity prediction		50	125
Post processing			
Max source/sink distance		50	125
Source/sink histogram threshold slack	0.5		
Max source/sink angle (degrees)	30		
Minimum path time (seconds)	2		
Time threshold to split opposite-direction track (seconds)	0.3		
Angle threshold to split opposite-direction track (degrees)	120		

Table 1. Test parameters.

4.1 User study

For the user study, 11 videos were used. These consisted of 6 unmodified videos and 5 modified videos with replayed cars that were detected using the proposed method. Each of the test videos was between 40 and 60 seconds long. The goal of this user study is to determine how the quality of custom videos compares to the “best possible” situation (namely the original videos). A total of 20 subjects have participated. Figure 22 shows an example of an unmodified and a modified video filmed at the same site.

Figure 22. An example frame of an unmodified video (top) and a modified one (bottom). You can see a car that is fading towards a spawn point (the horizon) in the zoomed-in box.

4.1.1 User study test setup

Before the test starts, subjects signed a consent form allowing them to quit at any point in the experiment without supplying a reason. Some extra information of subjects was also obtained. This included age, education, job, VR experience, problems with eyesight, problems with hearing. There was no plan to use any of this information, but it was noted regardless in case some correlation became apparent.

Next, a short explanation of the old VCNS was given. Subjects were only told that the old VCNS has been improved by the use of video and that the experiment was to determine the quality of this improvement. This means that during the experiment, they were unaware that some videos also contain replayed cars. After that, an explanation of what was expected of the subject was given. First a "control" video would be shown. This is just an unmodified video, and subject were told that it would count as the "perfect" situation. Then, a series of 10 videos (5 unmodified and 5 modified)

were to be shown in a random order, and subjects were told to give a score of 1 to 7 for each video, where 7 means as good or better as the control, and anything lower meant that something (or more than one thing) bothered them in the video. Subjects were free to make up their own scale between 1-7, since giving them an absolute worst-case would give away the car replays too much. They were simply told that a 1 should be completely unacceptable quality in their opinion. An example frame for each of the videos can be found in Figure 23. In order to test the end use-case of the VCNS a virtual plane flyover was included in each test, but subjects were told to just judge the visual quality. If a subject gave a score lower than a 7, they were encouraged to explain what caused the less-than-perfect situation, and their comments were written down.

Figure 23. Example frame for each of the user study videos. The top one is from the example video. The middle row contains the original videos. The final row contains the modified videos. As you can see, more than one video was shot at several location.

Subjects were shown an example of a stitching error, and were told to ignore these specific, static shifts in the world. This was done because the stitching errors were more obvious in some videos and less in others, and we did not want subject to judge them since they are not a part of the actual desired quality information. See Figure 24 for the particular example that was shown.

Figure 24. Stitching error example shown to subjects.

After that, a proper user-specific setup of the Oculus Rift DK2 was done. Subjects were told, before donning the HMD, that they should take it off as soon as they felt uncomfortable (dizzy, nauseous etc.) to prevents further issues. None of the subject experienced problems however, as the stationary 360-video experience is generally not nausea inducing since the virtual “character” does not move. Next, the Oculus calibration tool was run for each subject. This is a small program used to measure the interpupillary distance of a person, which is in turn used to compensate in the rendering software to prevent blur. This ensures the best possible experience for the user.

Finally, the actual VCNS program was started. When no video was playing, subjects would see the simple empty horizon that exists in an empty Unity3D scene. Before the videos were shown, a quick summary of the scoring and a reminder to not include stitching errors was given. Then, headphones were donned, and the actual experiment started. First, the control video was shown, and users were told to remember the quality as much as possible. Then, the other 10 videos were shown in random order and after each video, the scene reverted to the horizon. Subject were asked for their score after each video, as well as any additional comments about the video. Each experiment took around 25 to 30 minutes (including post-experiment discussion), depending on whether the subject had more questions or not.

4.1.2 User study results

Figure 25 shows the number of times a certain score was given to the modified and unmodified videos and Figure 26 shows the average score per video. Little over half of the unmodified videos were identified as such (score 7), namely 58%. Note that the number of scores happens to coincide with the percentages. This is an unplanned coincidence, since we had 20 participants, each giving out 5 scores for modified/unmodified videos respectively adding up to 100 scores per type of video. The unmodified videos received only 4% of scores lower than a 5. The scores for modified videos are spread out a little more, with 19% getting a 7 and only 8% being scored lower than 4. The scores 4-7 got a roughly equal number of votes. When we just look at the scores, we can see that the modified videos generally get lower scores, but in many cases they are not unacceptably bad.

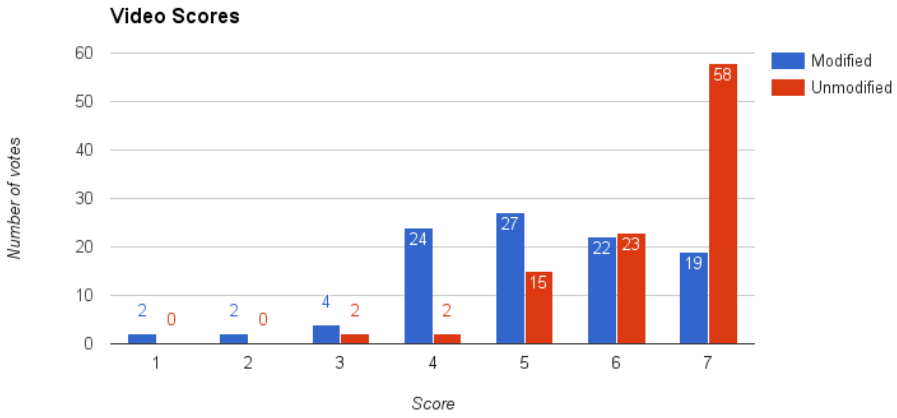

Figure 25. Distribution of scores given to the test videos, for both the modified and unmodified videos.

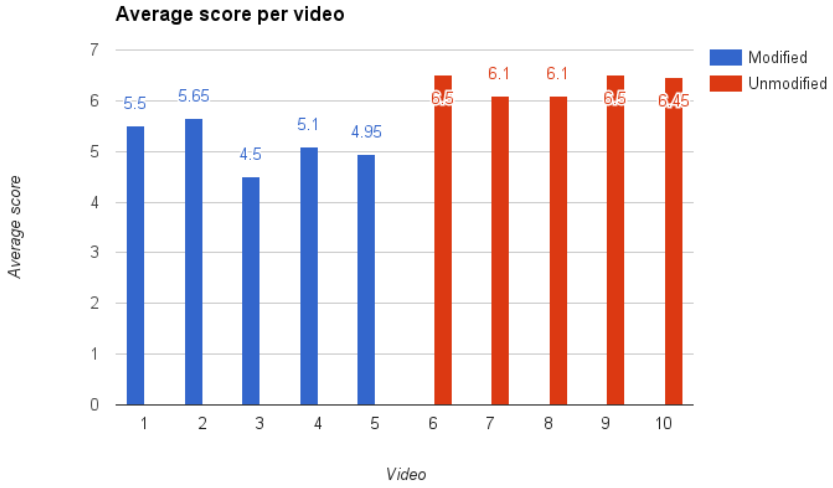

Figure 26. Average score per video.

To determine whether or not the modified videos are scored lower than the unmodified ones, a one-tailed independent samples t-test ($\alpha = 0.01$) was conducted to compare video scores of the modified and unmodified videos. One-tailed was chosen because we can already clearly see from the graph that the modified videos won't score significantly higher than unmodified ones. The result of this test was that there was indeed a significant difference in the scores for modified (Mean: 5.14, StdDeviation: 1.37) and unmodified (Mean: 6.33, StdDeviation: 0.94) videos; t(176)=-7.15, p(1.11E-11). This means that the modified videos are definitely scored lower than the unmodified ones.

Some more interesting information can be extracted from the comments that subjects gave when scoring a video lower than 7. For instance, out of the 81 scores lower than 7 for modified videos, only two third (54) were explicitly due to issues with the modified vehicles. The rest were other issues with the particular video unrelated to the replayed cars and were generally common issues throughout all the videos. These non-modification related issues (which were the same issues that were commented on when unmodified videos were given a lower score) are mostly any of three different issues. The most noticed issue is that of stuttery car movement. This problem is the result of a mismatch between video frame rate (60) and the refresh rate of the HMD (75), causing stutter in the video once every 4 frames. This issue was also present in the example video, but since it was the first video, most subjects did not notice it there. This unfortunate circumstance was not detected until there was not enough time left to process all videos again. The second issue was that of low resolution, either due to the low resolution of the DK2 or because of the video, or both. The third was the problem with wrong scale. Because we don't film with a stereoscopic setup, some sense of scale will be lost during stitching since there is no parallax effect. This results in things seeming too close or too far, and it can also make it appear as if the camera was way higher than the average person, even though it was placed at eye-height in the majority of the videos. Figure 27 shows a partial frame of a scene where such scale were most apparent to users. Since the edges of the balcony and the object around the user were rather close by, the lack of parallax is more noticeable than scenes without close objects. The contrast between close and far away parts of the video also cause bad stitching issues like in Figure 24, again mostly due to parallax.

Figure 27. A scene that causes scale issues. Users especially noticed that the broom was too big.

While these three problems were present in all videos, subjects appeared to become more critical to them after having watched more videos. For most subjects, it was the first time they had ever experienced any modern VR HMD, or any VR HMD at all, which may have led to them being in slight awe for the first few videos. As they became more accustomed to the experience, perhaps they started to focus more on the actual content. Unfortunately, we did not log the order in which each person was shown the videos, so there is no solid information on this. Some subjects did comment that they became uncertain about whether or not they saw certain issues in earlier videos as the experiment went on.

Two other issues were mentioned once or twice. One was that the airplane did not interact perfectly with the environment since it sometimes disappeared too soon when approaching a pole. This is a problem caused by slight inaccuracies in the manual mask for the foreground, and the issue was only noticed by two subjects who were focusing on the the airplane more than we wanted (but that is part of the freedom of subject to interpret 'visual quality'). The second issue was that some videos were shot on a cloudy day while others were shot on a sunny day. The comment was that the cloudy ones looked less nice and less lively, which is something that the subjects in question deemed part of the visual quality as well.

Another thing we noted was how some subjects did not notice any problems with cars, even when a vehicle that they were looking straight at showed some obvious artifacts. After the test we showed them the problem that they had not commented on, and they confirmed that they did in fact not notice it. This problem could maybe be attributed to the first-time HMD usage of the subject, or the general low resolution of the HMD.

The final thing we mention is that there were some outliers amongst the subjects' scores. Most noteworthy were two subjects. One(the one the most critical scores), who is responsible for both '3' scores for unmodified videos, as well as both '1' scores for modified ones. Strangely enough, this subject also gave a modified video a perfect 7. The other interesting subject scored every single video with a 7. This person did not notice any issues, and as mentioned above, confirmed that no

problems were noticed during the experiment after showing the issues in some the modified videos. Figure 28 shows the average score per user, where the two outliers are clearly visible (users 11 and 20).

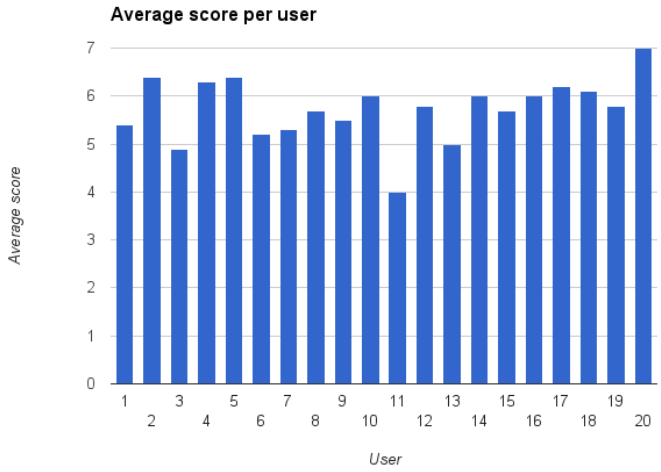

Figure 28. Average score per user.

In conclusion, it appears that the artifacts that are present in modified videos are definitely noticeable. Whether or not they stand out can vary wildly from user to user, and watching many videos in a row could make it more likely for someone to notice problems. In the future, videos should be resampled to match the frame rate of the desired headset before processing it with the proposed method to avoid the stuttering issue.

4.2 Objective study

For the objective study, several cars have been manually segmented by creating binary masks for specific frames of those cars. The goals is to determine how accurate the segmentation of the proposed method is.

4.2.1 Objective study test setup

The objective study has been performed using all 15 test videos. In every test video, 5 cars have been manually segmented totalling 75 manually segmented cars. The 5 cars per video were chosen by picking the ones that appeared closest to 5 timestamps in the video. These 5 timestamps are distributed evenly across the video. The stamps were calculated by taking the length of the video (minus 5 seconds at the start), setting the first timestamp at the start of the interval, and then taking $1/5^{th}$ of the total length between each next timestamp. The 5 second grace period that we remove at the start is there to give the background subtraction some time to adjust, as well as to eliminate some noise caused by shaking of the camera when giving the synchronization signal for stitching. Figure 29 shows a visual representation. As an example, imagine that a video is 105 seconds long. First, we get rid of the 5 seconds at the start, leaving us with 100 seconds. We calculate that $1/5^{th}$ of 100 seconds is 20 seconds, which will be our time between timestamps. So, the 5 timestamps in the video at which we look for appearing cars are at the 5, 25, 45, 65 and 85 second marks.

The specific cars were chosen using the timestamps at which they appeared. When calculating the timestamps to extract a car from, a 5-second grace period at the start of the video was ignored. This

was done to give the background subtraction some time to adjust, as well as to eliminate some noise caused by shaking of the camera when giving the synchronization signal for stitching. The remaining length of the video was divided by 5, and whichever car appeared closest to the start of each segment of 1/5th the length was chosen.

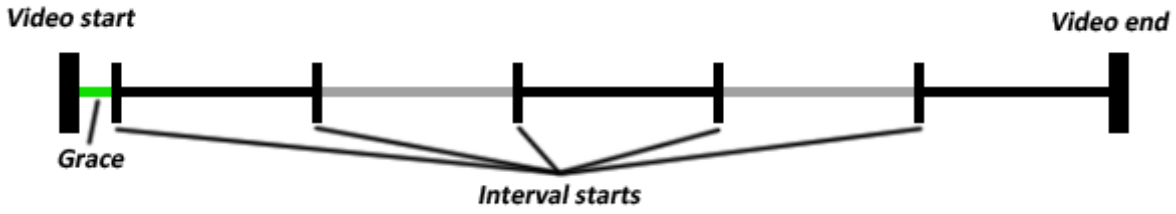

Figure 29. Time intervals for manual segmentation.

From the each of the videos (which are at 60 fps), every 60th frame was extracted (frame 1, frame 61, frame 121 etc). This means that the number of frames that is processed per car varies, depending on how long the car was in the video. For each car that was chosen for manual segmentation, every extracted frame that contained the car in question was processed. This processing consisted of creating a binary mask containing the pixels that should contain the car (as shown in Figure 30), if we ignore other cars that may be occluding the car in question. This means that not only the currently visible pixels were segmented, but also pixels that may have been covered by a non-static objects such as another car. Of course, the segmentation of these non-visible pixels is not 100% accurate since the necessary information is just not there, but we need these pixels to rate the accuracy of the interpolated frames in the proposed method. We also determined how many separate trackers a particular car is expected to have (by determining how many source/sink inducing locations the car passes).

Figure 30. Manually segmented car. The part of the road under the car is also considered part of the car, since it contains the reflection and shadow of the car. These two aspects need to be detected too in order to realistically replay the car later.

Next, we determined the trackers that tracks at least some part of a manually segmented car. If a tracker was found to track a manually segmented car, we consider the entire source/sink string of trackers to be linked to the car. With these links we can start comparing trackers to manual segmentations. We process each manual segmentation by taking the corresponding frames in the linked trackers, and applying operations to them to determine how well they match. When using the tracked frames the frames were used without the added blur to the mask, as the blur is only there

for added visual quality and has nothing to do with detection. For the tracked frames that did not have a manually segmented equivalent, the manual segmentation was assumed to be empty (all background). Likewise, for any manually segmented frame without a tracked equivalent, the tracked frames were considered empty. For the tracked frames, we use a binary mask that represents all the pixels that were part of the frame.

For every manually segmented frame for a car, an empty image was created. Every tracked frame that was linked to the manually segmented frame was compounded into the empty image by using the AND operator. This means that we obtain a single image of the size of the original video, which contains contains all the frame data that was detected for that car at that frame. We only consider the mask for the detected frames, since we do not incorporate color information in our measures. This full detected mask is then compared to the manually segmented frame. The amount of true positive (TP), false positive (FP) and false negative (FN) pixels were determined. Note that the amount of true negatives was omitted, since that particular measure has little meaning in our case. The absolute total number of TP/FP/FN can vary wildly per video and per car since they can have very different sizes, but the results will only discuss ratios so this does not matter in our case. We also determined the number of trackers for the car, as well as the number of source/sink strings.

4.2.2 Objective study results

While the user study showed some acceptable results since users did not find the visible glitches too troublesome, keep in mind that those videos only used cars that were fairly well-detected in our eyes. Many detections were rejected because they were obviously unacceptable, so we expect the objective results to tell a different story. Let's first discuss the results of the TP, FP and FN analysis. Figure 31 shows the mean and +1/-1 standard deviation of the recall (TP / (TP+FN)) and precision (TP / (TP+FP)) of all cars. Figure 32 shows the individual results for these measures per car.

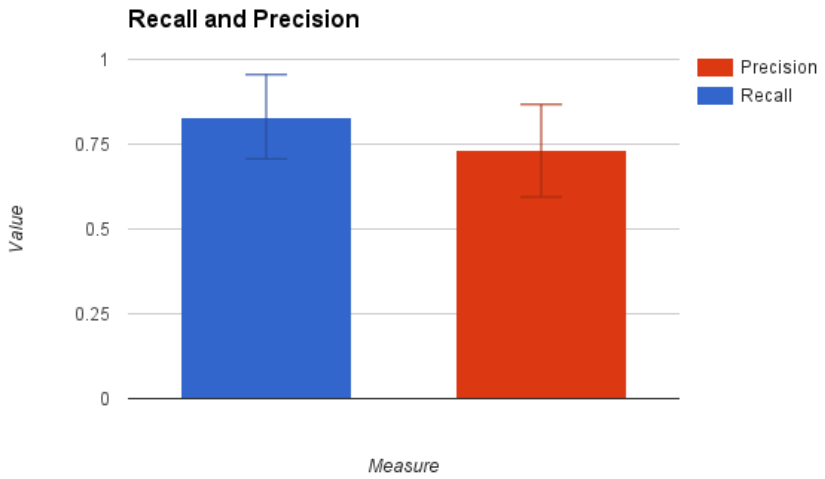

Figure 31. Mean and +1/-1 standard deviation of the recall and precision.

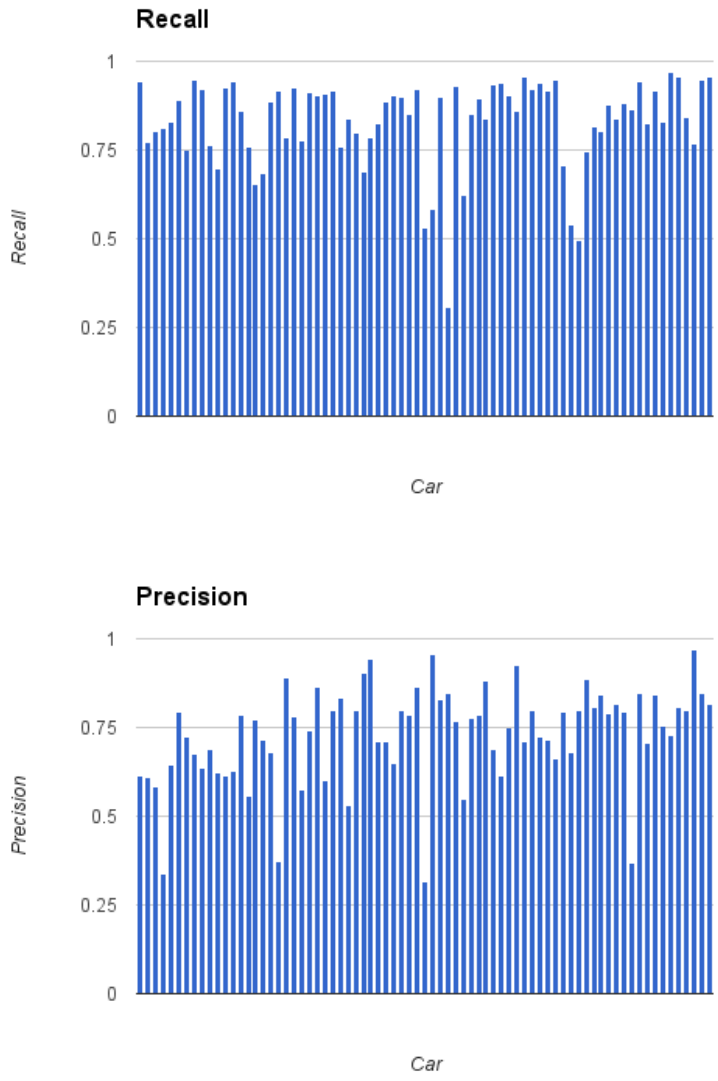

Figure 32. Recall and precision per car.

Recall measures how many of the pixels of the car have been detected by its tracker(s). On average, 85% of the pixels are detected with the majority staying above 75%, which is a respectable amount. However, since we use the binary mask for the measures, we lack color information. This means that while an interpolated piece of a tracker may have accurately predicted the position of many of the pixels, the color will still be slightly (and sometimes very) different from what the actual colors should be. Precision measures the amount of positive detections that are actually positives compared to the total number of detections. With an average precision of 73%, we see that quite a substantial number of extra pixels (27% of the total) are detected as if they were part of a car. This is not completely unsurprising though, as the method specifically aims to at least detect the car pixels, and as such has been tuned to overestimate sooner than it would underestimate. The primary reason for this overestimation is the influence of the closing operation, which especially affects vehicles with concave shapes. Then there are some outliers (for instance, the 4 sub-50% ones in the graph). These instances are generally caused by the source/sink strings containing trackers that track completely different cars, by the tackers missing entire frames of the car, and/or by a large overestimation due to the closing operation. Missing frames or invalid cars are then fully counted

towards the measures in every checked frame, which can add up quickly. Figures 33, 34 and 35 show a few situations that cause low objective scores.

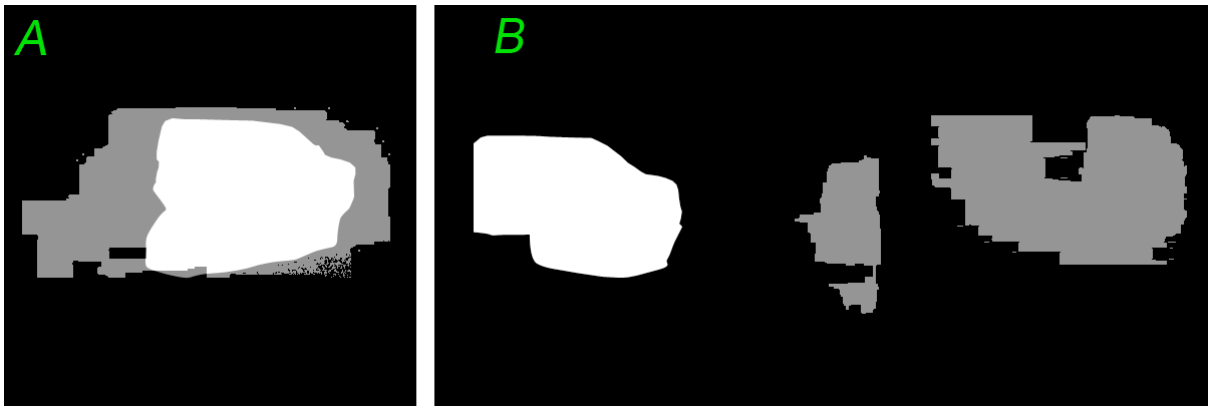

Figure 33. Problems that cause low precision scores. White shows the manual segmentation, grey show the detection. A) Severe overestimation due to interpolation. B) Tracking something unrelated to the car (and missing the car frame entirely).

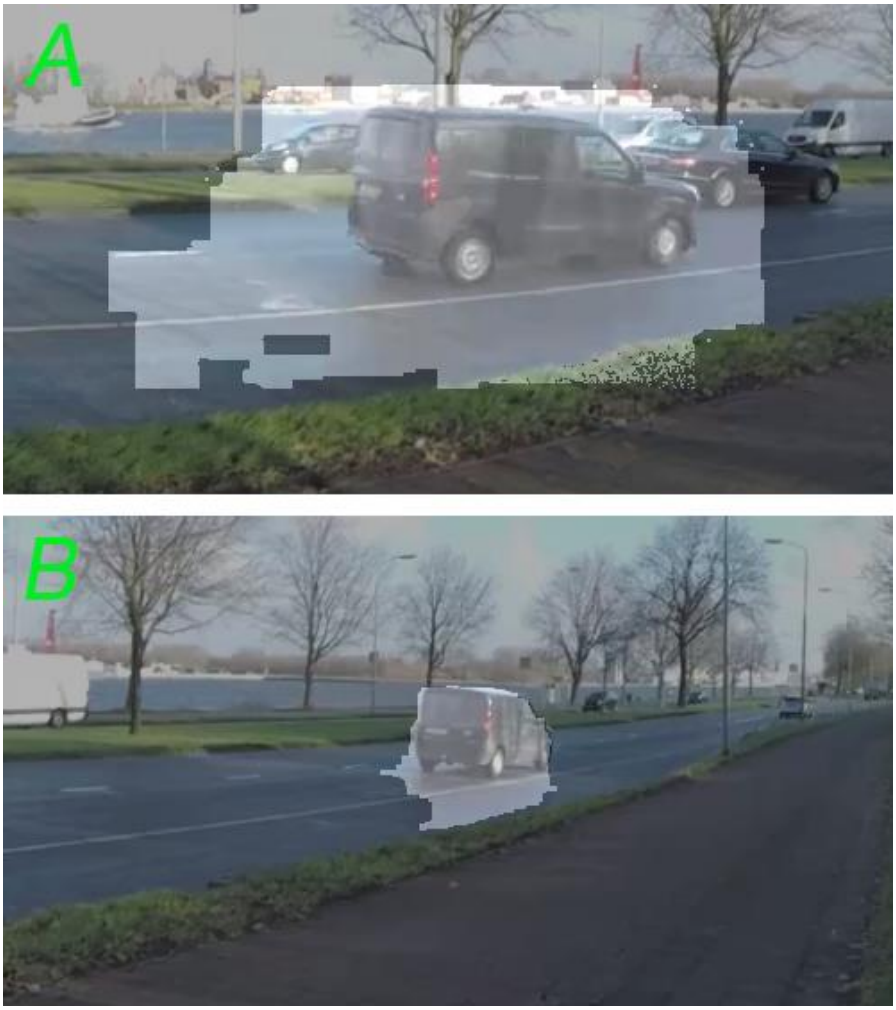

Figure 34. A car and its corresponding detection in two frames. The white overlay shows the detection (the surrounding area has been darkened for added contrast). A) A problem with interpolation caused by intersecting cars. B) A second later, the detection is fine.

Figure 35. Several examples of issues that hurt the objective scores. The left image shows a cleaner that was walking by, whose partial body is detected because it was in the foreground mask. The lower part of his legs are not. The center shows a car detection where a few frames contained a secondary car (the main detection is from the car going to the right). The right image shows a problem where two separate cars are linked as a source/sink pair even though they should be completely separate detections.

Next, we will discuss the number of trackers per car. Figure 36 shows the mean and +1/-1 standard deviation of the average number of source/sink strings per car, as well as the average ratio between the actual trackers and the expected trackers per car. Figure 37 shows these measure per car.

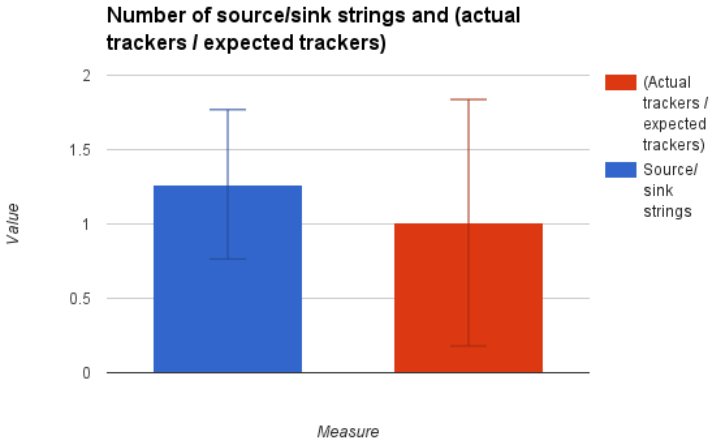

Figure 36. Mean and +1/-1 standard deviation of the number of source/sink strings, and the ratio (actual trackers / expected trackers).

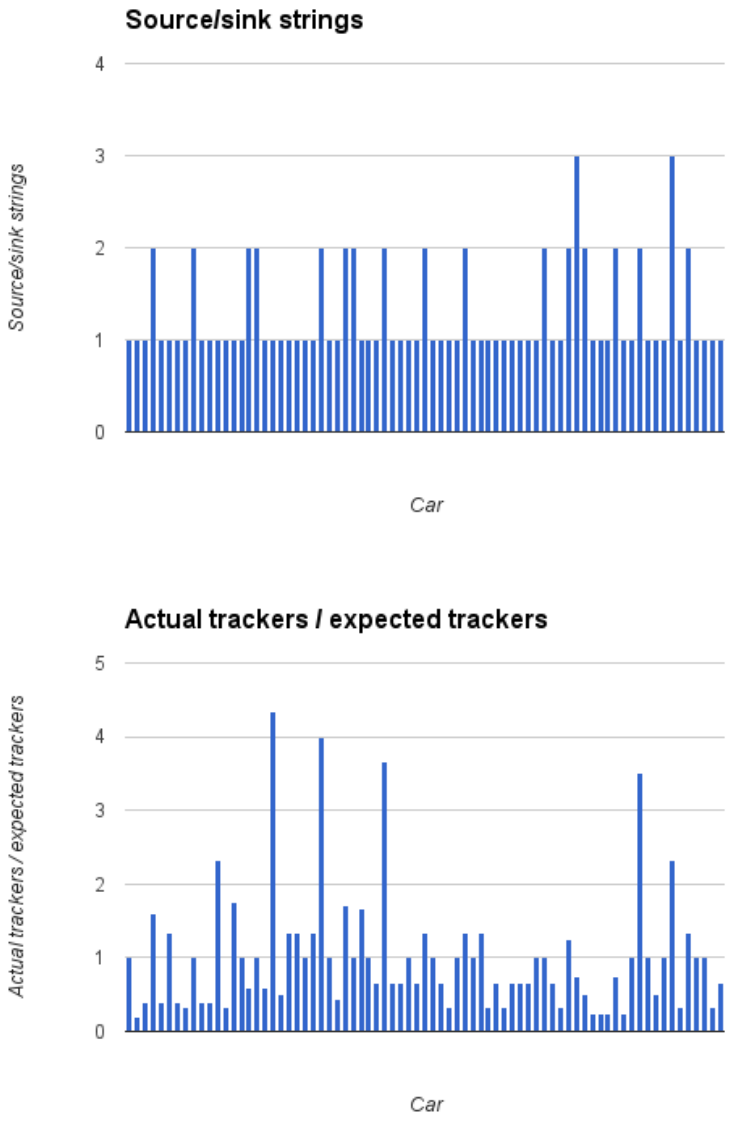

Figure 37. Number of source/sink strings and the ratio (actual trackers / expected trackers) per car.

When we look at the number of source/sink strings, the perfect scenario is a single string per car. 76% of the cars indeed had one string, slightly over 23% had two strings and slightly under 3% had 3 strings. There are two scenarios where more than one string can occur. The least bad of the two is when the two strings exclusively track the same car. In this case, an unfortunate set of circumstances have caused the two strings to not be linked together. Perhaps the car was not visible for an extended period of time, or perhaps a lot of cars crossed each other, causing unstable detection for a long part of the car's path. The second scenario is worse, namely when two strings partially track the same car, but one string also tracks a significant portion of a different car. In this case, it is likely that both strings are completely unusable.

The ratio of actual trackers compared to expected trackers should be 1 in a perfect scenario. When we look at the graph however, we can see that a mere 25% actually attains this number. The rest of the values vary wildly, which explains the large standard deviation (the standard deviation is almost as large as the mean). From this we can conclude that the number of trackers is completely

inaccurate compared to what is expected. Since the values vary so wildly, we theorize that even the cars that have the perfect ‘1’ ratio most likely do not have the source/sink relationships in the right places. A decent majority of wrong cases are the result of poles/trees not causing a source/sink situation. This is caused by the closing operation, which fills in the thin line that the pole creates in the foreground mask, and hence the pole is seen as part of the car. Some other cases occur when two similar cars are driving behind each other. If the first car creates a new tracker instead of using its old tracker because the new connected component was not properly matched, the post processing steps might link the two cars. This happens because a “new” car spawns and that car is similar to the second car, which is our definition of source/sink relationships. Source/sink relationships are not exclusive in our method, so a single tracker might have multiple sources. This means that the first car likely still gets the right source/sink relationship, but the second car also qualified and hence also gets it. further frame-by-frame study of each and every source/sink relationship would be required to get to exact the bottom of the issue, but this study has not been performed yet since it is rather intractable to manually inspect the large number of such relationships.

In conclusion, the objective quality of the proposed method is not as good as the user study would suggest. While most frames of the cars are detected with good accuracy, it can sometimes take just one visually obvious glitch to render the car unusable. For instance, interpolated frames can make it look like the car expands quickly, kind of like a balloon, if the frames that were interpolated between have a major difference in size. Another example of a no-go glitch is if a secondary car was part of the connected component of the first car during a few frames, which mean a secondary car would show up in several frames of the replay of the first car. So even if the objective measures say the car is decently segmented, it can still be unusable.

While a filmed video of about 3 minutes generally yields enough good detections to create an acceptable custom scenario, a fairly significant number of trackers are still unusable for such custom videos even if they don’t have a single obvious failure as mentioned above. An unusable detection can have various reasons, including but not limited to:

- The tracker(s) missed a number of the frames of the car, which means the car either appears somewhere in the middle of its path out of nowhere, or it disappears while it should have still been clearly visible. The individual frames themselves could also miss parts of the car due to inaccurate foreground detection.

- The source/sink strings also contained trackers that do not actually track the car we want. This could be fixed by the user by manually removing said source/sink relationship however.

- The quality of the interpolated frames are of low quality. This is often caused by the fact that the occlusion interval was too long and therefore the linear interpolation interpolates between two wildly different frames. Another problematic situation is when foreground objects contaminate the interpolation. For instance, when the closing operation causes a thin pole in front of the car to be part of a frame of the car, and that frame is used for interpolation, the pole will be interpolated along with it. This causes a copy of the part of the pole that overlapped the car to move along with the car, as if it was glued on.

5. Conclusion

We have proposed a method for automatic detection, tracking and segmentation of vehicles in panoramic equirectangular 360-degree videos. Working with 360-degree videos makes some aspects of tracking and segmentation harder. For instance, objects that are close to the camera exhibit larger changes in size than regular 2D video. Another issue is that cars often move towards a visible horizon, which makes them undetectably small. This does not happen in regular 2D video unless the camera is pointed nearly parallel to the road. Effort was made to have the method handle these situations.

The proposed method works by first doing a single front to back pass over all video frames. A GMM based foreground detection method is employed to detect moving objects. Multiple operations are used to refine the detected foreground, and connected component analysis is used to detect interesting objects. These objects are then tracked throughout the video. Once every frame has been processed, the trackers are examined further during post processing to filter out wrong detections. To obtain the segmented cars, little manual work is required since the segmentation method works autonomously once parameters are set up. Parameters for a certain type of scenario (for instance, cars passing by at ~10 meters away) only have to be determined once. They can then be reused for similar scenarios, often without further adjustments for individual videos. Before the program starts, the user can optionally spend roughly a minute to paint a mask for the background subtraction and click on the potential spawn points. The bulk of the manual work comes later, in the form of validation by a user to ensure that a detection is usable and assigning a lane to the detections in order to maintain proper depth-ordering when replaying the cars. The user also has to choose when he wants cars to appear in the custom videos.

In order to evaluate the method, two studies were performed. A user study used a group of 20 test subjects to determine whether or not the custom videos were significantly worse from the original videos. This study showed that while there is a measurable difference in quality, the vast majority of subject do not deem the custom videos unacceptable. The second study was an objective study, meant to measure the objective accuracy of the tracked cars by comparing detection results to manual segmentations. This study showed that the objective accuracy of the detection is not perfect by any means, since there is a significant number of detections that have low precision (and sometimes recall) scores. The low precision often means that multiple cars are detected as being part of a single car, which then leads to the detection being unusable since a random extra car popping in a few frames up during a video is very apparent. The user study and objective study appear to give some conflicting results, with the user study saying acceptable while the objective study says the results are poor on average. This is because the detections that were used to create videos for the user study have been hand-selected by a human. During this process, the large amount of poor detections that cause the low average objective scores were simply not selected and therefore users only got to see the best-detected cars. Even so, the cars used in the custom videos often still contain some smaller glitches, such as missing pixels or missing frames near the spawnpoint/endpoint of the car. However, user testing has shown that while these quirks do result in lower perceived quality of the custom videos than the original videos, the custom videos still are of acceptable quality.

6. Future work

There is a large number of different changes that can be made to the method to potentially increase the quality of the results. For the online portion, an obvious candidate for improvement is the background subtraction method. While the current GMM-based method is acceptable, there are some newer and better methods out there such as ViBe [18]. Improving the background subtraction means the refinement operations such as opening and closing can be set less aggressive, or even be omitted completely. It can also help with the detection of larger vehicles such as trucks.

The object model used in the online portion (currently consisting of position/size/color information) for the trackers can also be improved. The first thing that could help is changing the color model from YcbCr to something else like HSV, and potentially not using all color channels in the histogram. Some more specific shape information instead of just the bounding box size could be taken into account as well, to improve matching between tracker and connected components, as well as source/sink matching. This could help, for instance, with a problem where a tracker that is currently tracking a sports car suddenly starts tracking an SUV (or more commonly in our method: gets a wrong source/sink relation with the SUV). This could greatly help filter out problems that occur when two cars' connected components merge. The current method for calculating the velocity and acceleration of the object could also be improved. Kalman filtering is a popular technique that can be tried out, as well as using higher order function for least-squares. These higher order functions could be useful, because while car movement is almost linear when the cars are far away, the panoramic effect of the 360-degree video can give the paths a significant curvature when they are close to the camera. A completely different approach could require the user to draw lines across the center of the roads, and use those to guide the velocity/acceleration computation. These lines could then double as a guide to automatically assign lanes to cars instead of assigning the manually (these lanes are required to ensure proper depth-ordering during car playback).

During post-processing, classification of objects can be a potential boon to the method. If we were able to classify whether or not a tracked car contains a single car vs multiple cars, we can reject merged trackers instead of having them be detected as a single car. Classification could also help when there are frames that are missing large portion of the vehicle, which tends to happen with long trucks. In this scenario we could perhaps detect that the classification algorithm is not detecting a complete vehicle in many frames, and decide to discard the detection. There is a problem with classification however, since a misclassification can actually be detrimental to performance of the method with no easy way to recover.

Post-processing could also benefit from better interpolation. Right now, the interpolation used for missing frames is the most basic linear interpolation possible. For each frame in the missing interval, a linear blend of the two closest non-interpolated frames before and after the current frame is used. This works fairly well for short intervals of a few frames, but fails when the intervals get longer, and gets downright ugly when the sizes of the two original frames differ significantly. A more content-aware interpolation method could help alleviate these issues. Finding matching points of reference on the original frames could allow us to interpolate more accurately by enforcing constraint to have these points move smoothly, instead of having them be static and just fading them out. Like with classification however, severe mismatches between these points of reference can render an interpolated series of frames unusable, if for instance a headlight gets interpolated to a tire.

More functionality for manual post processing is also a possible improvement. It can happen that a single car is detected across multiple trackers. Right now, there is no easy way of linking the trackers together manually. Some functionality to allow user to define such links would then allow the program to automatically create the necessary source/sink relationships at the right time, and potentially do some interpolation for missing frames.

Finally, another large point of improvement lies with the parameters. Currently, the method has a large amount of parameters, which makes finding the right set difficult. Perhaps some parameters can be automatically calculated by analog input, such as the user drawing a line across the thickest pole to determine the maximum source/sink distance. Perhaps some parameters have so little influence that they can be omitted. Automatic global parameter optimisation (for instance, simulated annealing) could be done using the manual segmentations that are available now, but this process would likely be prohibitively lengthy because of the running time of the algorithm and the number of parameters.

While this method was primarily developed for car detection, the fact that it does not explicitly classify cars allows it to track other objects as well. During our tests, we had a few decently tracked scooters, as well as cyclists and even a pedestrian or two. The problem with these three is that interpolation does not work as well, especially with pedestrians, since they are less rigid. The method does not work well with many inter-object occlusions, so any form of pedestrian tracking in even slightly crowded areas is not possible. Within the NLR, we are looking to possibly track airplanes as well. The problem with that however, is that airplanes move rather slowly in the video, and as such the background subtraction might struggle, since it will start seeing the plane as background. It is probably possible to adjust the parameters in such a way that slow moving objects don't get added to the background, but we are unsure how well this will work with the current GMM application. A new background subtraction method might be required for this. If airplanes can be properly tracked, then a possibly further application of this method is to replay the airplane, but with simulated sound. This could then be used to create an absolutely realistic video, which can be combined with the state of the art sound simulation of the VCNS to for instance show what the sound would be like if the exact same plane flies by with more silent engines. With the current hardware however, such an application would only be useful if the plane is relatively low in the sky, because in that case the difference between a CG plane and an original plane is hardly noticeable. But once even higher resolution video and HMDs are used, such an application might find use. Higher resolution cameras (and potentially lossless recording) could also help in detection, since lower resolutions in compressed videos sometimes contains problematic blocks of pixels. In this case part of a car might simply not even be there in the actual video, as we found out during manual segmentation of the cars. Even worse, sometimes very small cars even completely disappeared for a frame. Better stitching of the videos can also help a great deal in overall quality in some scenarios where static objects are close to the camera, since the seams are currently rather apparent in this situation. Since VR is quickly becoming a booming industry, several specialized 360-degree cameras are already entering the market, which will likely make stitching much easier.

References

[1] Y. Xin, J. Hou, L. Dong, L. Ding: *A self-adaptive optical flow method for the moving object detection in the video sequences*. Optik - International Journal for Light and Electron Optics, Volume 125, Issue 19, Pages 5690–5694, 2014.

[2] C. Stauffer, W.E.L. Grimson: *Adaptive background mixture models for real-time tracking*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 2, 1999.

[3] Z. Zivkovic: *Improved Adaptive Gaussian Mixture Model for Background Subtraction*. Proceedings of the 17th International Conference on Pattern Recognition, Volume 2, Pages 28-31, 2004.

[4] W. Zeng, W. Gao, D. Zhao: *Automatic moving object extraction in mpeg video*. Proceedings of the 2003 International Symposium on Circuits and Systems, Volume 2, Pages II-524 – II-527, 2003.

[5] T. Meier, K.N. Ngan: *Automatic segmentation of moving objects for video object plane generation*. IEEE Transactions on Circuits and Systems for Video Technology, Volume 8, Issue 5, Pages 525 – 538, 1998.

[6] R. Xiaobo, T.X. Han, Z. He: *Ensemble Video Object Cut in Highly Dynamic Scenes*. IEEE Conference on Computer Vision and Pattern Recognition, Pages 1947 – 1954, 2013.

[7] D. Comaniciu, V. Ramesh, P. Meer: *Kernel-Based Object Tracking*. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 25, Issue 5, Pages 564 – 577, 2003.

[8] K. Deguchi, O. Kawanaka, T. Okatani: *Object tracking by the mean-shift of regional color distribution combined with the particle-filter algorithm*. Proceedings of the 17th International Conference on Pattern Recognition, Volume 3, Pages 506 -509, 2004.

[9] Z.H. Khan, I.Y.H. Gu, A.G. Backhouse: *Robust Visual Object Tracking Using Multi-Mode Anisotropic Mean Shift and Particle Filters*. IEEE Transactions on Circuits and Systems for Video Technology, Volume 21, Issue 1, Pages 74 – 87, 2011.

[10] C. Beyan, A. Temizel: *Adaptive mean-shift for automated multi object tracking*. Computer Vision, IET, Volume 6, Pages 1 – 12, Issue 1, 2012.

[11] J.S. Kim, D.H. Yeom, Y.H. Joo: *Fast and Robust Algorithm of Tracking Multiple Moving Objects for Intelligent Video Surveillance Systems*. IEEE Transactions on Consumer Electronics, Volume 57, Issue 3, Pages 1165 – 1170, 2011.

[12] R. Cucchiara, C. Grana, G. Neri, M. Piccardi, A. Prati: *The Sakbot system for moving object detection and tracking*. Video-Based Surveillance Systems, Springer US, Pages 145 – 157.

[13] L. Zhang, L. van der Maarten: *Structure Preserving Object Tracking*. IEEE Conference on Computer Vision and Pattern Recognition, Pages 1838 – 1845, 2013.

[14] A. Papazoglou, V. Ferrari: *Fast Object Segmentation in Unconstrained Video*. International Conference on Computer Vision, Pages 1777 – 1784, 2013.

[15] X. Bai, J. Wang, D. Simons, G. Sapiro: *Video SnapCut robust video object cutout using localized classifiers*. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH, Volume 28, Issue 3, Article No. 70, 2009.

[16] M. Kim, J.G. Jeon, J.S. Kwak, M.H. Lee, C. Ahn: *Moving object segmentation in video sequences by user interaction and automatic object tracking*. Image and Vision Computing, Volume 19, Issue 5, Pages 245 -260, 2001.

[17] I. Endres, D. Hoiem: *Category Independent Object Proposals*. Lecture Notes in Computer Science, Volume 6315, Page 575 – 588, 2010.

[18] O. Barnich, M. van Droogenbroeck: *A universal background subtraction algorithm for video sequences*. IEEE Transactions on Image Processing, Volume 20, Issue 6, Pages 1709 – 1724, 2010.

[19] https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game

[20] X. Ren, J. Malik: *Learning a classification model for segmentation*. Proceedings of the ninth IEEE International Conference on Computer Vision, Volume 1, Pages 10 – 17, 2003.

[21] A.G.A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, H. Wensheng: *Multi-Object Tracking Through Simultaneous Long Occlusions and Split-Merge Conditions*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Volume 1, Pages 666-673, 2006.