LiveLayer: Real-time Traffic Video Visualisation on Geographical Maps

Simon Walton, Min Chen and David Ebert

(a) (b) (c) (d)

Fig. 1. Traffic videos *(a)* are projected onto a dynamic layer, called LiveLayer, superimposed on top of satellite imagery *(b)*. A visual mapping scheme provides additional visualisations of traffic measurements *((c) and (d))*.

Abstract— While video cameras are an ubiquitous and intuitive means of collecting traffic information, it has been difficult to utilise such an information source for creating a global view in real time traffic visualisation. In this paper, we present a real-time method for transforming video information in quasi-3D to spatial information in a 2D planar visualisation, which acts as a live layer on top of a conventional satellite image-based mapping system. To facilitate rapid and reliable camera-map calibration, we make use of a semi-automatic calibration scheme. The scheme allows users to pre-define essential projection attributes for each camera in a set-up stage, and map these video streams onto a live layer of the map. In this paper, we also describe several schemes for visually mapping information extracted from videos, including traffic speed and density, and uncertainty in illustration. This form of traffic video visualisation can potentially reduce the need for specialised traffic sensory devices and infrastructure, and enable better utilisation of the existing video-based traffic sensory network.

Index Terms—Multimedia (Image/Video/Music) Visualization, User Interfaces, Uncertainty Visualization, Visual Design, Geographic/Geospatial Visualization

1 Introduction

In recent years, map visualisation with real-time traffic information has become widely available on the Internet (e.g., SigAlert, Google Maps). The information is typically collected by a traffic counting infrastructure comprising thousands of traffic flow monitoring sensors. Commonly sensors include fibre-optic, piezo electric and inductive loops, infrared, laser and Bluetooth-based sensors, and weight-in-motion (bending plate, pneumatic tube). While these types of sensors provide relatively accurate information about numbers of vehicles, their deployment is usually restricted by the shortcomings of each type of sensor. For instance, weight-in-motion sensors and loop sensors require intrusive installation on the road, and are difficult to deploy in ungated open space (e.g., some large car parks) [23]. Above-ground optical and electromagnetic sensors (excluding videos) have limited signal ranging, and are erroneous in measuring complex traffic flows. In addition, they confine real-time traffic visualisation to a limited set of visual designs, typically colour-coded lines or textual annotation on maps.

We propose to use video information to enrich real-time traffic visualisation. Since video cameras have been used extensively for traffic monitoring (e.g., some 1400 traffic cameras in the UK), these could add a significant amount of information if they were utilised for real-time traffic visualisation. Furthermore, videos can provide a much richer visual experience, allowing users to observe traffic in imagery information, in addition to textual or colour-coded numbers. In this paper, we present a system for the real-time visualisation of streaming traffic video, and provide a family of techniques for addressing various challenges; including a more complex design space for different foci (i.e., imagery, vehicle counts, and uncertainty) (see Sections 5 and 8); geometric mapping from video space in quasi-3D to geographical space in 2D (see Section 6); and information filtering (see Section 7). We have given a particular emphasis on the practical usability of the proposed techniques, and developed a prototype system to demonstrate their feasibility without relying on any assumption of the existence of a fully automatic computer vision system for camera calibration that has often proven to be unreliable and onerous in practice. Instead, we developed a graphical user interface which allows each camera to be calibrated in no more than a few minutes.

- *Simon Walton and Min Chen are with Swansea University, E-mail s.j.walton@swansea.ac.uk, m.chen@swansea.ac.uk.*
- *David Ebert is with Purdue University, E-mail ebertd@ecn.purdue.edu.*

Manuscript received 31 March 2011; accepted 1 August 2011; posted online 23 October 2011; mailed on 14 October 2011.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

2 Related Work

There are many types of sensory device for collecting real-time road traffic information. Hummady provided an excellent set of comments on the shortcomings of each type of sensory devices in [13], and also proposed the use of video cameras as data sources for road traffic management. His proposal relies primarily on a computational device that is capable of analysing the scene and recognising 'vehicles, pedestrians, 2-wheel vehicles, etc.' automatically. The proposal does not include any detailed algorithmic solution to the traffic video analysis, which remains to be a very challenging technical problem in computer vision; nor does it involve the use of any visualisation techniques.

There has been a huge amount of effort in developing techniques for scene analysis and object recognition in the context of traffic video processing. Zhu and Li [35] used a simple threshold-based background model for tracking moving vehicles, combined with smoothing filters. Vibha *et al.* [31] gave a vehicle-counting system, based on counting connected components of the background mask (also using a

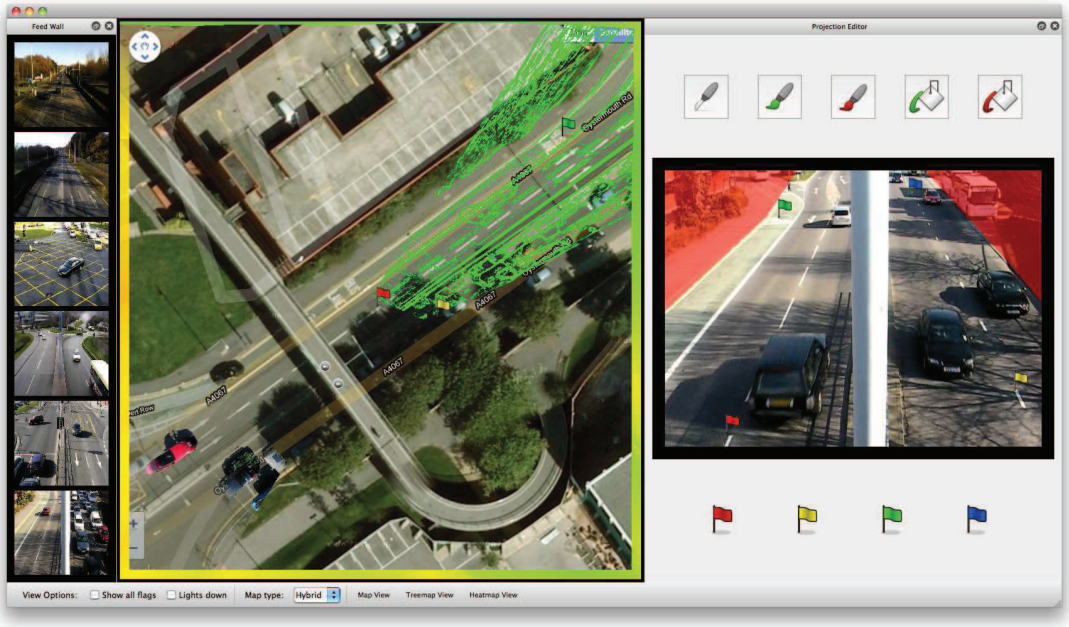


Fig. 2. The LiveLayer User Interface. *(left)*: the feed wall containing camera feeds ready for definition; *(middle)*: the main map view for live projection renderings; *(right)*: the projection editor for defining the projection parameters.

threshold-based background model). Using background models such as Gaussian mixture models to survey moving objects presents issues with the learning rate adopted that governs how quickly the model reacts to newly-background areas of the image. With a slower learning rate, moving objects tend to leave a trail of pixels that are struggling to learn about the background left in the objects' wake. Cheung and Kamath [8] used a combination of a slower-learning background model combined with blob-tracking to remove these trails. Many specialised algorithms have also been developed, such as for shadow detection [15], object tracking [22], and identification of ground-plane homographies [3].

However, the state of the art frameworks rely heavily on 'two types of a priori information: 1) the contextual information of the camera's field of view and 2) sets of predefined behaviour scenarios' [18]. As semantic specifications at different cameras vary significantly, it is difficult to port an automatic solution from a laboratory camera to a large number of real world cameras. Because it is problematic (or at least time-consuming) to specify such semantic information for each individual camera, using automatic computer vision to gather information from videos is yet to become a practical solution. Due to this reason, this work takes an approach that does not rely on automatic scene analysis and object recognition. Instead, we provide an easy-to-use user interface for entering necessary contextual information, and generate dynamic visualisations without involving object and event recognition.

The most commonly-used convention for road traffic visualisation is to colour-code the lines or areas that represent roads on a map (e.g., [28]). Other traditional visual forms, such as time series plots and heatmaps (without using geographical maps) have also been used to visualise traffic information (e.g., [20]). Ang *et al.* [2] gave a visual analytical approach to traffic surveillance from multiple cameras with feature extraction to estimate vehicle trajectories. Andrienko and Andrienko [1] also described a visual analytical approach to visualising large amounts of movement data by aggregating and clustering the data to present on a geographical map as colour-coded arrows.

Video visualisation was first proposed by Daniel and Chen [9] as a means for summarising a video segment into a single static image. This subject was further studied by Chen *et al.* [7] who confirmed that ordinary users can learn to detect and recognise *visual signatures* of events from video visualisation. Wang *et al.* [32] proposed to combine videos with a 3D environment model to support situational understanding. Remero *et al.* [27] investigated the visualisation of activities captured by overhead video cameras in natural settings. Botchen *et al.* [5] presented a visual design called Video Perpetue-Gram (VPG), and Höferlin *et al.* [12] applied VPGs to the visualisation of sports videos.

Cartography is a discipline with an amazing history. Many influential cartographers, including Robinson [26] Bertin [4] MacEachren [21] have provided a comprehensive collection of visual and semiotic guidelines. While these guidelines remain hugely valuable to map visualisation, the modern data sources, such as videos, have introduced new challenges to real-time information capturing, dynamic visualisation and image-based visual design. This work represents a serious attempt to bring maps alive with video information.

3 System Overview

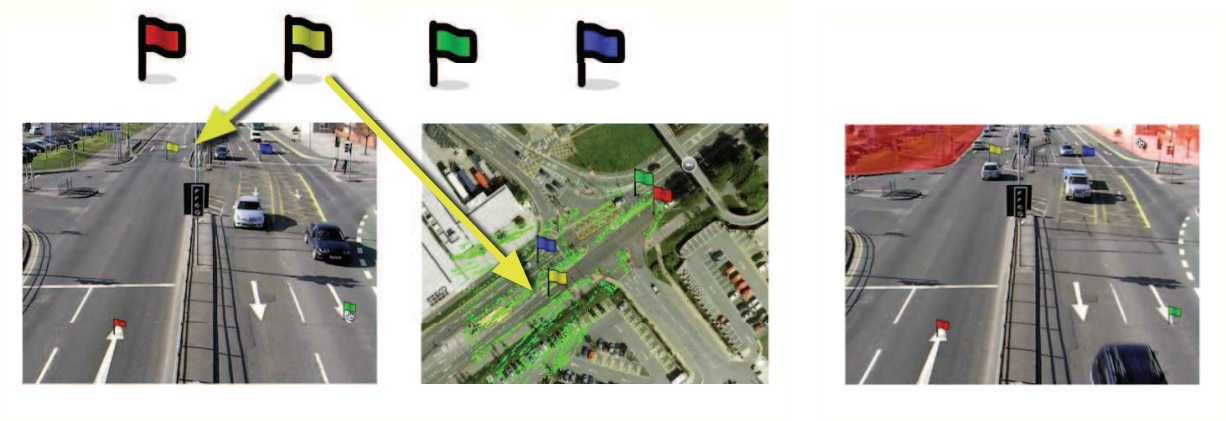
Creating a live layer of traffic visualisation from videos requires us to address several technical challenges:

1. A video typically captures the perspective projection of a real-world scene, which is referred to as quasi-3D. A live layer is a 2D plane, with imagery information corresponding to the video. It is therefore necessary to provide a cost-effective means of specifying this transformation, and also to remove non-road information from the video.

2. It is necessary to remove the non-traffic (or *background*) information on the road.

3. The video information needs to be transformed from quasi-3D to 2D and rendered onto a map in real-time.

4. It is necessary to provide a means to combine traffic information from different cameras onto a single live layer.



(a) User drags the *flags* onto the video *(left)* and map *(right)* to match locations

(b) User *cuts* non-road areas out of the video

Fig. 3. Defining a new camera, step-by-step: the *Flag & Cut* process.

5. In some cases, it is desirable to visualise traffic information in a more abstract form. It is thus necessary to provide an efficient method for mapping imagery information to measurements to be visualised.

The technical solutions to these challenges will be discussed in detail in Sections 4–8. Our system provides a graphical user interface that allows the user to define the projections for a number of video streams onto their corresponding coverage areas on a geographical map, provides real-time rendering of the projected video streams (incorporating a series of visual design strategies), and additionally assists in visualising the camera network as a whole. These features are integrated into one piece of software for the ease of proving our concept.

The graphical user interface for our system (pictured in Fig. 2) was developed using Qt 4.7 on Mac OS X. OpenCV is used for various computational and imaging aspects of the system, and OpenGL / GLSL is used for all of the visualisation output. For the mapping imagery and associated semantic functionality, we use Google Maps (embedded in a QWebFrame) and the Google Maps API.

3.1 Acquiring Test Videos

To test the system, a number of traffic videos were acquired from the surrounding area using a digital camcorder recording video at 640×480 resolution, at six to twelve frames per second. Since the system presented here concentrates on the mechanics of defining the projections and using the outputs of these projections, all test videos used were re-encoded as MJPEG to reduce the computational cost of decoding. The test videos were mostly acquired from footbridges using a 'bendy' tripod to attach the camera to the bridge. The average height of these bridges is around 20ft, and thus there is much intra-vehicle occlusion in each scene; particularly as the vehicles travel further from the camera and the height of the vehicle becomes a large factor. The vast majority of bridges utilised were of metal construction and thus suffered from vibration / wobble due to wind conditions and also pedestrians walking on other parts of the structure. In addition, conditions during the filming of many videos were fairly windy and sometimes wet. The combination of these factors resulted in a noticeable wobble in most videos, which we welcomed as it allows us to judge our visualisation strategy under unfavourable (and realistic) conditions.

It should be noted that our system currently focuses only on stationary traffic cameras – that is, cameras without pan / tilt functionality.

4 Defining a Camera's Projection

Kong *et al.* [17] gave a set of algorithms for extracting road boundaries from an image containing a vanishing point, but the algorithms unfortunately rely on the existence of a vanishing point in the image, which we choose not to rely on given the wide variety of tilt angles witnessed with local traffic cameras. He *et al.* [11] gave a promising road detection system that utilises colour information and user-provided camera parameters for classification, but the authors admit that it is difficult to design an algorithm that works well for all types of road. Generally, computer vision techniques that rely on discovering such distinct features are heavily reliant upon the camera angle and image quality. In a system such as ours that is aimed at utilising existing camera networks, not only are we unlikely to be dealing with simple roads that have well-defined edges and distant endpoints, but additional factors such as adverse weather conditions, harsh shadows, and various unrelated scene objects ensure that automatic attempts to define what a 'road' actually is are unlikely to succeed.

Our approach utilises human intelligence to define how the streaming video from each camera corresponds to its associated projection on the map. The two aspects of this definition are:

- The *homography* of the system that defines the mapping between video space and map space; and
- The *mask* that defines the areas of the video that should be included in the final rendering.

A previous implementation of our system made use of road 'templates' that could be selected to place predefined road layouts onto the video for the user to manipulate. These roads were defined as closed non-convex polygons, and effectively defined the homography through their vertices and also the final mask through their internal area. During testing, we found this system to be unnecessarily difficult and restricting due to the sheer variety of road widths, layouts, and camera angles in the real world. Due to these issues, a much simpler and intuitive system that we name *Flag & Cut* was developed. Fig. 3 gives a conceptual overview of this process, and the next sections describe the process of defining the homography and defining the mask, respectively.

4.1 Point Correspondence through Flags

The user first initiates the definition of a new camera to the system by locating its feed on the feed wall (left, Fig. 2) and dragging it into the main map view, where the system places the camera on the map (represented by a small camera icon). The feed wall acts a simple container for available video streams. Once a camera is dropped, the *projection editor* (right, Fig. 2) opens for the user to begin the process of defining this camera's homography.

We use a point correspondence system to compute the homography (more details in Section 6). The points are shown to the user as small flags: one red; one yellow; one green; and one blue. It is relatively trivial for the user to learn to find places on the video image that

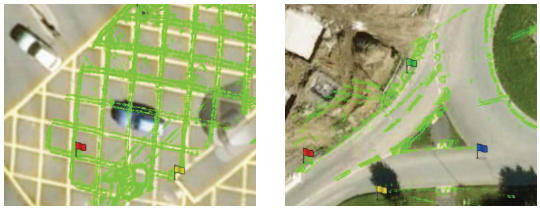


Fig. 4. A Sobel filter is applied to the projected video as the user moves the flags, assisting with correct alignment against real-world markers.

correspond with places on the satellite image given in the main map view. We have found that placemarkers such as lane arrows, box junction hatches, signposts, lampposts and other numerous road markings provide useful flag positions. There is the possibility of there being discrepancies in the video image obtained from the camera and the satellite imagery provided on the map, usually due to out-of-date satellite imagery or obstructions. In such cases where the opportunities for placemarker-based flag placement are difficult, the user relies more on intuition to converge to a suitable result.

Once the user has defined four flags on the video and four flags on the map, the system automatically computes the homography and begins rendering the live projected video. The user is free to drag the flags around at any time, and during dragging, a Sobel filter is applied to the live rendering output (see Fig. 4) so that correspondences between edge-based features in the video and a features on the satellite imagery can be judged more easily.

4.2 Cutting the Roads

With the homography computed, the user can begin to define which portions of the video are road, and which are non-road. By 'non-road', we imply regions of the input video that provide no benefit when projected onto the map (pavements, sky, etc). We provide a series of tools for this, and they are listed below.

The *scalpel* tool allows for polygonal regions of the video to be cut away by clicking to draw a series of lines through the video, intersecting with two boundary edges of the video. The user can define which side of the slice to discard by hovering over the video with the mouse and selecting the correct candidate (please see the demo video for a demonstration). If the user prefers, they can also perform exactly the same cut on the map projection using the computed projection boundaries, or cut inner regions unattached to the border.

The *brush* tool behaves similarly to the brush tool most are familiar with in basic image manipulation software, except in our case it can either define areas as road (*green*, pictured) or non-road (*red*). The user selects a brush size and then brushes onto either the video or the map view. Upon brushing onto the map, the circular shape of the brush is correctly warped onto the video. Working on the map is sometimes preferable as it allows for more precise movements for distant regions.

The *floodfill* tool again behaves similarly to the floodfill functionality in basic image manipulation software. The *green* bucket defines areas as road, and *red* (pictured) as non-road. This tool is useful for quickly undoing mistakes in the mask, as the mask is strictly binary and thus does not suffer from aliasing difficulties. As with the other tools, this tool can operate either on the video or on the map projection.

As the user is defining the mask using these tools, the live projection continues rendering with the updated mask for instant feedback: the video displays a red overlay over non-road areas of the image (see Fig. 3), while the live projection on the map shows the entire perspective-warped live video with non-road areas removed to show the satellite imagery underneath. The process of defining a camera is iterative in nature, as the user can modify the flags and mask at any time.

5 Visual Design of Background Model

In projecting the live video onto the map (minus 'non-road' parts of the user mask), the projection of the road surface and associated 'background' pixels becomes distracting (particularly at high distance where interpolation becomes dominant) and creates a harsh transition at the boundary edges. Differences between the satellite imagery obtained from the mapping service and the projected area from the video cause more distractions also. In addition, we wish to identify pixels likely belonging to vehicles for additional visualisation strategies later. Therefore, we decided upon using a background model to identify foreground pixels (moving vehicles). Our system employs a background model based on a Gaussian mixture model (GMM) as first proposed for background removal by Friedman and Russell [10]. These models have since been improved and importantly implemented on the GPU using first conventional shaders [19] and then CUDA [25]; this applicability to the GPU is a beneficial property when considering future scope for our system.

Section 5.1 first details our approach to utilising information from the Gaussian mixture model for the purposes of ascertaining the certainty of the model's accuracy, and 5.2 details our rendering strategy that utilises this information.

5.1 Uncertainty in the Background Model

Any background model has inherent uncertainty, with the definition of 'certainty' tied to the context of the scene. For urban traffic video where traffic frequently comes to a halt (e.g. at a set of traffic lights), it is not acceptable for such traffic to be absorbed into the background after a certain amount of time. Our aim therefore is to accept this uncertainty and to introduce methods that ensure that temporarily-stationary vehicles on the live projection do not absorb into the background. Additionally, we choose not to remove shadows cast by vehicles as they provide an important visual cue that is reflected in one viewing traffic in real life.

Assume we have a video pixel position p situated at a point where traffic regularly comes to a standstill for minutes at a time. We refer to a Gaussian mixture model cluster's time in a particular position as its *maturity*. If a car moves into p and stops, p will eventually become part of the background as it begins to outweigh its clustered rivals. Setting a smaller learning rate can alleviate this somewhat, but at the expense of slower convergence elsewhere. If p's topmost cluster changes rapidly, then this probably indicates movement in the area. When its cluster becomes more mature, we become less certain as time goes on whether it represents the road or whether it represents a stationary car; thus, the certainty is linked to the cluster's maturity. Additionally, if the approximate colour of the road in the scene $road_{rgb}$ is known, then we can increase our certainty that p is background with p's colour distance to this road colour. Clamping a cluster's maturity to $[0,1]$, the certainty for p is given in our system as in Equation 1 where $tc(p)$ gives the top cluster for pixel p.

$$certainty(p) = 1 - (min(|| p_{rgb} - road_{rgb} ||^2, 1) * tc(p)_{maturity}) \quad (1)$$

Equation 1 can now be used to build a per-frame 2D *certainty field*.

5.2 Combining the Background Model with Histogram Matching

There are many examples in the literature of histogram information being extracted from video to improve the guesses made by a background model [16, 29]. Our approach is to embrace the uncertainty of background models, ensuring that the model's misjudgements do not result in information loss for the user. We achieve this by giving more prominence to pixels that have a high certainty of being foreground, and less prominence to pixels that have a lower certainty of being foreground. Using our strategy, projected pixels, including stationary objects, will always be visible to the user.

Before rendering, a limited-domain histogram matching process that matches the limited colour range of the road in the input video to the colour of the road on the satellite imagery occurs, outputting a

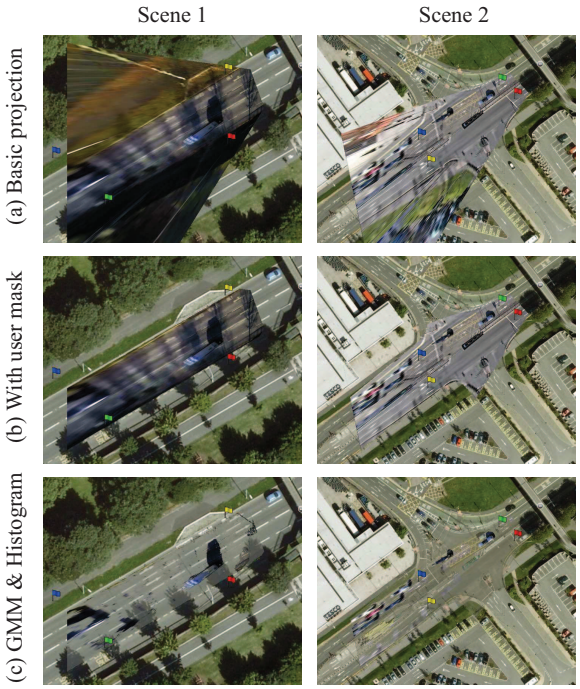


Fig. 5. A comparison of rendering two scenes with three different rendering modes. (a) shows the basic projection of the video onto the map; (b) shows the projection with the user mask enabled, cutting away non-road parts of the scene; (c) shows our Gaussian mixture model & histogram matching method.

histogram lookup table (LUT) that can be used to transform the road colour found in the video to the road colour found on the map. The process of computing this table can be found later in Section 6.2. Assume we have a pixel p in video-space related to some point in map space. Available to this pixel are three attributes:

- The *user mask* as described in Section 4.2: a binary mask where 0 corresponds to non-road and 1 corresponds to road;
- The *certainty field* as described in Section 5.1: with values in the range $[0, 1]$ where 0 is 'uncertain' and 1 is 'certain';
- The *foreground mask* obtained from the Gaussian Mixture Model: a binary mask where 0 corresponds to 'background' and 1 corresponds to 'foreground'.

The system determines the certainty of p being a foreground pixel by multiplying the binary foreground mask obtained from the Gaussian mixture model with the $[0, 1]$ certainty field to obtain a final foreground pixel certainty f_p. Given this certainty and the colour of the video at p, we can give the following output colours and opacities:

Foreground pixels : full opacity, with the colour defined as the original *RGB* values obtained from the video stream at p;

Background pixels : half opacity, with the colour defined as the colour of the *RGB* values obtained from the video stream at p found in the histogram LUT.

This method *blends* the colour of the road given in the input video with the colour of the road on the satellite image using a histogram matching strategy. Since the histogram LUT matches only a limited colour range (video road colour $\rightarrow$ map road colour), the histogram lookup function $h(p)$ also performs one additional check: if the LUT is empty at the lookup position for all colour components (and thus is not part of the source histogram's domain), then the original colour

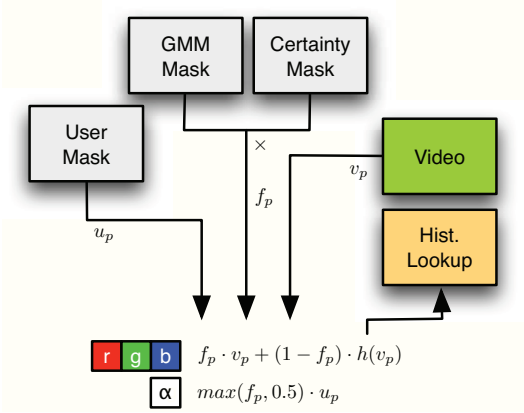


Fig. 6. A pixel p's final fragment colour is decided upon based on the binary user mask, the Gaussian mixture model binary mask, the $[0, 1]$ certainty mask, the video frame, and the histogram lookup table.

is returned unchanged. The effect is that a stationary car will still be visible. Even if the car is of similar colour to the road and thus goes through the histogram LUT, then the car is still visible (though not as clearly).

6 Geometric Mapping & Rendering

6.1 Homography Computations

Given the point correspondences provided by the user in the flag definition stage, the homography between video space and map space can be calculated. We use OpenCV's camera calibration functions to compute the homography matrices, which require a minimum of four point correspondences. We first ascertain the overall boundary of the projected video onto the map, given the current point correspondences. Given a point v_{xy} inside the normalised video dimensions, we wish to find a matrix Hb that gives us the $m_{lat,lng}$ corresponding coordinates on the map:

$$\begin{bmatrix} m_{lat} \\ m_{lng} \\ 1 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ 1 \end{bmatrix} \begin{bmatrix} Hb_{11} & Hb_{12} & Hb_{13} \\ Hb_{21} & Hb_{22} & Hb_{23} \\ Hb_{31} & Hb_{32} & Hb_{33} \end{bmatrix}$$

Once Hb is computed, the corner points of the video are projected to the map to obtain an initial projection (lat, lng) bounding box. This bounding box will initially be unworkably large due to the nature of the perspective projection – some points inevitably will lay outside of the projected plane and into infinite space. In an ideal scenario where the user has placed a camera in a location and the exact semantic information about the road's trajectory at that point is known, it would be possible to detect the vanishing point by projecting two parallel lines representing the road boundaries in map space back to video space and detecting the intersection of these lines. We should however not rely on such semantic information being available, and instead present a more general solution.

Using Hb, each pixel position within the video's dimensions is projected to the map, and its resulting (lat, lng) coordinates are stored in a 2D field. This field is then traversed, row by column, computing the approximate real-world distances (in metres) between each position and its immediate north & west pixel neighbours. If the distance between two pixels projected onto the map is below one meter in both directions, then the pixel is included in the projection boundary computation; otherwise, it is discarded. This has the effect of setting a cut-off point in the distance before the horizon is reached, and also serves to set a empirically-tested threshold on the projected video's resolution. Once this projection boundary is known, two further homographies are computed: H to project from video space to map space (within its newly-computed boundaries) and its inverse H^{-1} to project from map space back into the video. Working in this normalised space

is ideal for GPUs, where normalised texture coordinates across the boundary can be multiplied by the mapping matrix to convert between spaces.

6.2 Histogram Computation

We now outline the process of calculating the cumulative density functions (CDFs) of the source and target images to be used in the histogram matching process. As discussed, the aim of our visual design is to match the colour of the road contained within the video to the colour of the road contained in the satellite imagery to provide a more aesthetically pleasing result and more importantly to reduce the chances of completely removing falsely-identified 'background' pixels.

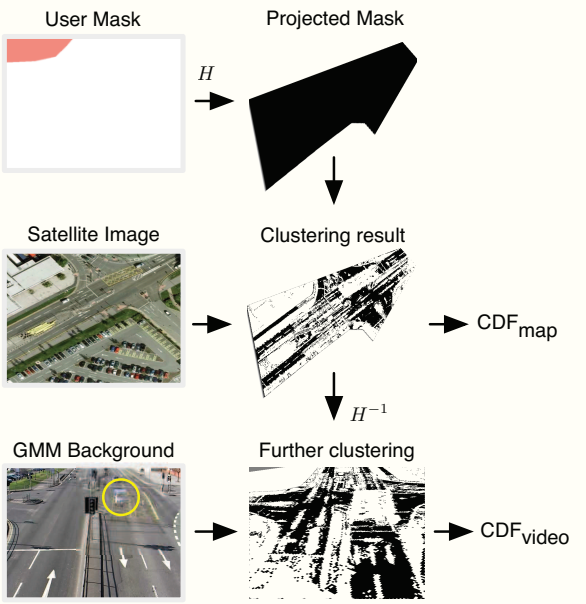


Fig. 7. Computing the histograms to be used for histogram matching. The user mask is first projected to the map, where a clustering operation computes the most likely road colour for the target CDF. After inverse-projecting back to the video, a final clustering is performed to obtain the source CDF.

Fig. 7 gives an overview of the CDF computation process for both the source and the target histograms. In this case, the source histogram is the histogram of the road contained in the video, and the target histogram is the histogram of the road contained in the satellite imagery. The task therefore becomes one of discovering which pixels of the video and which pixels of the satellite imagery are likely those of the road surface. In the satellite imagery used by our system, conditions are generally good with limited harsh shadows caused by a low sun, and similar hues between roads. There are however frequently cars present on the road which makes any naive approaches (such as using the projection of the user mask over the map and computing the histogram) difficult.

6.2.1 Cluster and Project Process

We begin by projecting the user mask onto the map using our homography matrix H (see Fig. 7 where the black pixels represent the 'road' area and white pixels represent the 'non-road' area). This projection gives a good working base for removing the worst of the unneeded scene regions. Next, we apply a clustering algorithm to the pixel colour values of the underlying satellite road surface to ascertain the most frequent colour cluster that is found on the scene. When a new pixel is matched to an existing cluster, that cluster's weight is biased according to the ratio of its RGB values:

$$weight(p_{rgb}) = (1 - (\mid p_r - p_g \mid)) \cdot (1 - (\mid p_b - (0.94 \cdot p_b \mid)) \quad (2)$$

Equation 2 gives a higher weight to pixels that have similar ratios of red to green, and a relative $\approx 6\%$ drop in the blue component. This 0.94 value has been found empirically by measuring the colour distributions of roads from the satellite imagery: the roads are generally grey but with a drop in blue due to the wavelength of the sun's light. There exist exceptions in certain areas: for example, special lanes painted different colours such as green bus lanes; but we typically find that these do not present a problem when weighed against the majority of the road.

Once the clustering algorithm has completed, we create a binary mask of those pixels on the map belonging to the cluster with the most evidence. Once this stage is completed, this mask is projected back onto the video using inverse homography matrix H^{-1}. From the Gaussian mixture model, we obtain a 'background' image by taking the current mean colour values of the top distribution for each pixel. Using this background image, a final clustering operation is performed on the video cluster mask using colour information from the GMM background. This final clustering step removes any erroneous information such as shadows being included in the CDF computation. Finally, an RGB histogram lookup table is computed using the source CDF and target CDF.

6.2.2 Histogram Update Frequency

In Fig. 7, it is evident that the background model has not been running for a long time as a blue car present at the beginning of the sequence (circled in yellow) has only partially absorbed into the background. In addition, changes in lighting levels and the overall colour temperature of the scene will eventually render the initial histogram inaccurate as time passes. For this reason, in addition to recalibrating the histogram upon changes to the homography, our system automatically updates the histogram for each camera every ten minutes, with an additional update scheduled for two minutes after the GMM begins computing the background model.

6.3 Rendering the Map View

For each camera whose projection boundaries intersect the view, the system first checks if it is necessary to obtain a new snapshot of the current map imagery based on changes to the viewing parameters or the completion status of the map's image tiles. If a new snapshot is obtained, it is provided to the histogram matching process and also uploaded to the GPU as a texture for later use. The camera's foreground mask (obtained from the Gaussian mixture model) is now uploaded as a 2D alpha texture, along with the certainty field, and the current camera frame as an RGB texture. The histogram lookup table obtained from the computations described in Section 6.2 is additionally provided to the fragment shader, encoded as a 1D RGB texture of width 256 pixels (for 24-bit colour space). Next, a framebuffer object is bound as the current render target, along with a custom fragment shader, and the map's projected boundaries are rendered as a quad with normalised texture coordinates.

There are two main fragment shaders: one to perform the Sobel filter on the live video (for when the user is dragging a flag), and the main fragment shader for rendering the scene using the background model and histogram information. In both cases, the inverse homography matrix H^{-1} is provided as a uniform variable and multiplied by the current texture coordinate (followed by a division by the w component) to obtain the final value inside the video frame texture. The fragment shader discards the current fragment if this texture coordinate falls outside the $[0,1]$ boundary, or if the user mask indicates that the sampled area is non-road. For the live video fragment shader, the blending operation shown in Fig. 6 is performed to obtain the final fragment. Once all cameras have been rendered to the framebuffer object, the satellite imagery texture is rendered to screen, and two separate fragment shaders perform a morphological closure operation on the framebuffer object before blending it to the screen.

6.4 Scalability: Multiresolution Fields

The GPU has no difficulties performing the relatively small number of matrix multiplications and blending operations for the inverse map-

ping, and the number of operations is bound by the pixel area of the map on screen. However, as more cameras are added to the system, the load increases – particularly on the CPU. This is mainly due to the requirement of more GMM computations (the most expensive operation) and more data (foreground mask, certainty field) being uploaded to the GPU. We observe that depending on the viewing parameters, some cameras take priority over others. In addition, if the user has zoomed out quite some distance to view a wide area, then calculating high-quality fields for each camera becomes less necessary as the resolution goes mostly unused. Although the focus of our work is not on performance (since a large factor will always be the decoding of multiple video feeds), and the software is multi-threaded, the usability of the system is severely hampered when this computational expense begins to deny the GUI thread its CPU cycles.

We created a statically-templated class that supports multiresolution fields of any type (i.e. unsigned char, Gaussian cluster pointers, etc). The fields are initalised with the full-scale field, and use only a subset of this allocated field for each resolution below full resolution. When a new resolution is requested, the object automatically scales the field up or down by copying the data blocks to the newly-requested resolution. A multiresolution field object is used for the Gaussian mixture model distributions, the resulting foreground mask, the certainty field, and additionally the video frames to be uploaded to the GPU. Since our software supports potentially many map view widgets containing potentially the same camera (due to the treemap system described later), multiresolution objects for a particular camera are scaled based on the highest resolution required for any views currently rendering that camera. The resolution is decided upon based on the highest zoom level and whether the camera active in any views. Higher zoom levels warrant higher resolutions, and cameras outside of any views warrant lower resolutions. One exception is that if the user is editing a homography, then that camera temporarily uses the highest-resolution fields to assist with flag placement.

When the user zooms into a camera's projection or drags the map to bring a camera into view, the foreground model may initially appear low-resolution as the cluster blocks have been duplicated to their neighbours, but gradually the system converges to a higher resolution; essentially imitating an adaptive refinement scheme. The objects additionally take care of correct texture uploads to the GPU, setting OpenGL's `GL_PACK_ROW_LENGTH` value accordingly.

7 Summarising the Camera Network

7.1 Obtaining Route Information

To allow for particular aspects of functionality relating to information on the physical location of roads, the system obtains route information from Google Maps' Directions API. Each time a new camera is added to the system, the system computes all viable routes between it and its nearest n neighbours, where n is set to 10 in our test system. We set this maximum to limit the potential exponential growth in routes. Requests for directions are grouped into camera pairs (to request a list of viable routes between two cameras) and are sent to Google DirectionsService asynchronously through Qt's QWebFrame JavaScript interactivity. The results from this request are bundled into a string to return to the application where they are unbundled from the string into structured objects.

Between two cameras, it is possible that more than one route will be suggested. Attached to each route, we record the *overview path* – a list of (lat, lng) coordinates representing a vector path of the route, and the *instructions* of the route – a list of coordinates for each specific instruction (e.g. 'turn left', etc). The overview path and the instructions of the route form roughly the same path, with the overview points giving a more finely detailed path that follows the road curvature. For use with the system's heatmap feature (see Section 8.3), we also store for each point on the overview path the instruction point that has occurred previously up to that point on the overview path.

7.1.1 Dealing with Two-Road Layouts

If we simply choose the camera positions as the user has defined them on the map as the start / end points for the routes, then difficulties can often occur where the projections are defined over two-road systems; that is, areas under the projection that are defined by the directions data as being two separate roads rather than one road over a number of lanes. Fig. 8 demonstrates this, where cam_a's projection (*green*) has been defined over two roads (*yellow*). When defining a start and end point for a route, the Directions API will snap any given points to the nearest road automatically. The danger is that the snapped position at cam_a will be on the wrong side of the road, giving a suboptimal route to cam_b as the route has to 'turn around' first or take an otherwise longer route (*red*) than the most optimal route on the correct side (*green*).

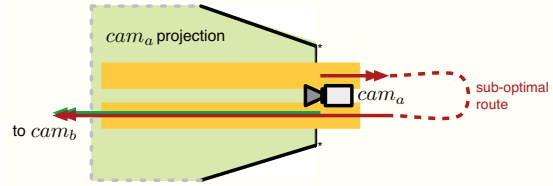


Fig. 8. Obtaining route information: the choice of exactly where to begin the directions request can have a large impact on the computed route.

To reduce this possibility, our algorithm chooses two different start / end locations per request, with the assumption that under the projection area there *could* exist a multi-road system that will negatively affect routing information. The choice of points is based upon the nearest left / right boundary of user's defined projection (marked with asterisks in Fig. 8). Four requests are therefore sent per $cam_a \rightarrow cam_b$ pair, and the set of routes with minimal total distance of the first route (which will always be the optimal result) is chosen as the correct set of routes for that camera pair. The system also records which side of the road the route begins on by checking incoming routes for disparities between their initial route vectors (the initial direction from the camera that the route takes). If the dot product of two of the initial vectors is negative, then we mark this camera as surveying a two-road system.

8 Visual Mapping

Many systems employing car-counting / density estimation algorithms exist, using methods ranging from correlation matching [31], to blobs [8], to more advanced methods such as hidden Markov models [30]. Most methods of estimating traffic density use computer vision or statistical methods for the aim of counting individual vehicles, and often overlook certainty in the outputs [14]. For a user to interpret this data in a real-time system, the data must be visualised with its certainty level to give intuitive and actionable information that can be utilised for discovering trends and patterns. Rather than attempting to automatically infer statistics on the incoming data at computational expense, we instead choose to augment the user's visual intelligence by utilising a visualisation that presents the data to the user in a summarised form, complete with certainly level.

8.1 Pixel Switch Algorithm

We have developed a visual mapping system that implements a conceptually straightforward pixel-based algorithm which maps directly to an intuitive visualisation that shows the user a real-time estimate of the traffic's speed (measured as the amount of *change* per frame – is the traffic fast-moving, or slow-moving?), density (is there much traffic on the road?), and certainty level (how certain are we that these facts are true?). Our algorithm can be viewed as a hybrid of a sensor-based and a video-based input: we treat each projected pixel in the scene as a switch, triggering whenever a car moves over it.

Algorithm 1 gives an overview of the per-frame 'pixel switch' method operating on a frame i. The algorithm also uses the previous frame $i-1$, the foreground mask from the background model, the certainty field, user mask. Once the *density*, *change*, and *certainty* values have been accumulated, they are normalised by dividing by the *total*. For a given time window, a *rolling mean* is calculated for a collection

Algorithm 1 The Pixel Switch algorithm for traffic measurements.

```
foreground masks f_i, f_{i-1}
certainty field c_i
user mask u
change ← density ← total ← 0
for all pixel indexes p in the projected map-space do
    p' ← p inverse mapped back to video-space
    if u(p') = road then
        if f_{i-1}(p') = background and f_i(p') = foreground then
            change ← change + 1
        end if
        if f_i(p') = foreground then
            density ← density + 1
        end if
        certainty ← certainty + c_i(p')
        total ← total + 1
    end if
end for
```

of normalised values of the same type (e.g., *density*) obtained from the series of consecutive frames in the window. The rolling means for *density*, *change*, and *certainty* are then passed onto the timeline and heatmap visualisation as detailed below. The uncertainty in this system can be classified as *validity uncertainty* – that is, the uncertainty is based on deductive inferences [24] rather than, for example, the quality of the data.

8.2 Treemap Dashboard

Treemaps have been applied to diverse hierarchal datasets, where the colour coding of individual cells is used to introduce various attributes of the datasets, and the sizes of each cell used to represent the size of each data block. Treemaps have been used to visualise geographical data, such as work by Wood *et al.* [34] where the treemap cells correlate with physical locations on a map; or work by Wood and Dykes [33] that presents spatially-ordered treemaps that modify the original squarified treemaps algorithm by Bruls *et al.* [6] to visually order the treemap cells according to their underlaying data's spatial position.

We wish to provide a simple means for the user to discover camera-covered regions in the system without the necessity of browsing through lists of cameras. In addition, we wish to incorporate our statistical modules into this system so that it is easy to identify, for example, the areas of the system with the densest traffic. To this end, we have developed the Treemap Dashboard (see Fig. 9(a)). The Treemap Dashboard gives an overview of the cameras in the system using a flat (single-level) treemap arrangement. Each cell in the treemap is itself a live map view with the same functionality and rendering capabilities as the main map view. The dashboard assists the user in discovering localised clusters of cameras in particular geographical areas (for example, a group of cameras around a busy junction, or two cameras covering different sides of a bridge).

To construct the treemap, we use a *k*-means clustering algorithm to group the cameras into a user-specifed maximum number of clusters (set to 5 by default) before sending these clusters to the squarify algorithm by Bruls *et al.* . Each cell in the treemap is centred around the mean position of the cameras in the cluster, with the view bounds set to the union of its camera's projection boundaries plus an additional 5% as padding. The user can interact with the map in the same manner as with the main map view, except that double-clicking the cell will revert to the main map view centred on that cluster.

8.2.1 Timeline Visualisation

Each cell of the Treemap Dashboard has a streaming representation of that cell's statistical history running around its perimeter, which we refer to as that cluster's *timeline*. This timeline is also visible in the normal map view (see Fig. 2), where it summarises all cameras that are currently in view as an average. Data in the timeline begins at the top left of the cell and continually streams clockwise around the perimeter of the cell, fading in opacity near the end to become invisible once it reaches the top left again. Our visual mapping of the cluster's state averages the results of all cameras' density, change and certainty values obtained from the pixel switch algorithm. For *density*, the timeline can be seen to act as a continuous chart, where the chart rises into the cell to represent denser traffic and falls back into the cell's boundary when the traffic density is low. For *speed* (the *change* value), we use a colour ramp where red indicates the slowest traffic and green indicates the fastest-moving traffic. This leaves *certainty*, which is encoded as opacity – the more certain we are of the values, the more opaque that area of the chart becomes; and when less certain, it becomes more transparent.

8.3 Heatmap Summarisation

In addition to the Treemap Dashboard, a heatmap view has been developed (Fig. 9(b)) that utilises the information gained from the Google Directions API to present a global view of traffic conditions, using a similar visual mapping to that of the timeline system. The heatmap view can be initiated manually using its toolbar button on the main interface; however, more usefully, this heatmap can also be gradually blended into view as the user zooms out of the map (as shown in Fig. 10). This allows for the same map view to act as a local and global information provider depending on the viewing parameters. The overall aim of the heatmap is to show the user an estimate of traffic conditions *at* cameras and also *between* cameras by utilising the route information between them. Using the heatmap view, the user can watch the colours of the heatmap change in real-time according to certainty-based traffic conditions between cameras.

Without data on the traffic conditions on the roads between cameras, there is some degree of uncertainty on such paths. If there is a high volume of traffic at cam_a, and no traffic at cam_b and just one road between the two, then we can assume with some certainty that the traffic on this road is also of high density. However, this certainty is reduced if we discover that there is, for example, a roundabout on that route. If there exists a route between two cameras cam_a and cam_b, and two instruction points (e.g. 'turn left') in the middle of this route, then after each instruction point along the path of the route, we multiply the initial certainty level by a reduction factor $0 < \mu < 1$. Making a sensible choice of the value would depend on the local geographical conditions. In this work, we set $\mu = .75$, which does not reflect any local geographical conditions, but serves to reduce the certainty effectively as the route crosses more instruction points towards its destination camera.

In a similar manner to the visual mapping in the previous section, we link certainty to opacity when rendering, but combine density and speed into one value before colour-coding using the same colour ramp (red: dense, slow; green: sparse, fast). In addition to this, we utilise information on whether the road system is multi-road if available (as calculated in Section 7.1). If the road is deemed multi-road, then we assume that the leftmost half of the video image represents mostly the left-hand side of the road, and the rightmost half the right-hand side. The pixel-switch algorithm can now split its outputs into these two halves, and the initial direction vectors of each route are matched to the correct side of the road to enhance accuracy in areas where one side of the road is busy and the other is quiet.

The rendering algorithm for the heatmap uses OpenGL point sprites to allow for the drawing of point primitives with texture coordinates generated across their boundaries. These texture coordinates are used to give the points a circular shape and opacity fade towards the edges of the circle. The rendering algorithm interpolates the density / change / certainty values of the cameras at either end of the route along the route path, reducing the certainty level when it detects that the route passes an instruction point. Each route segment is broken into a number of point sprites, one rendered on top of another, and each point sprite has the current density / change encoded into its colour / alpha components. The blending operation is set to the standard 'over' operator to accumulate the point sprites into a continuous 'streak' of varying colour and opacity across the route.

(a) Treemap Dashboard (b) Heatmap

Fig. 9. Visual Mapping using (a) the Treemap Dashboard and (b) the heatmap. The *Treemap Dashboard* allows for a greater understanding of the camera network through a clustering and visual mapping strategy. The perimeter graph around each cluster's cell models traffic density using graph thickness, the speed of traffic with colour, and the certainty of these values with opacity. The *heatmap* provides a geographical overview of traffic conditions based on the potential routes between cameras using a similar visual mapping scheme.

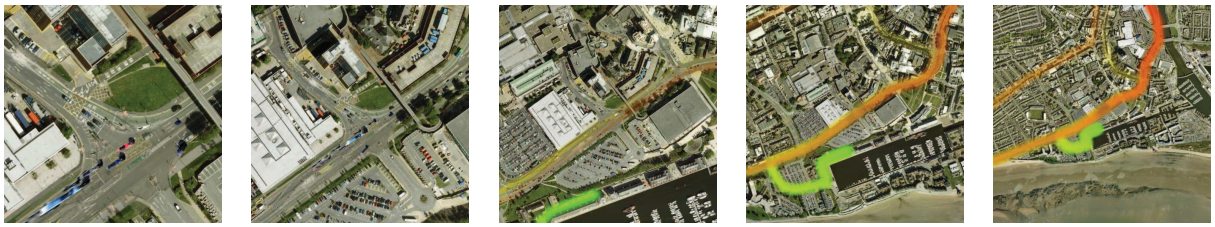


Fig. 10. The user zooms out of the main map view, and the system fades the heatmap gradually into view to compensate for the reduced resolution of the live renderings.

9 Discussion & Future Work

There is much scope for future work in this area. For the homography specification, currently, the method relies solely on the video and satellite imagery providing the same landmarks for registration. If these are not consistent due to out-of-date imagery or occlusions then the user must use intuition and iterative trial-and-error to obtain a good result. Ideally there is another data source for resolving such inconsistency. In addition, small changes in flag placement further away from the camera have larger consequences for the projection, so a method for helping the user make more precise cursor positioning on the video is required. Our histogram-based matching has been tested only with good road conditions, and we would be interested to see how our matching strategy could evolve to cope with different road colours and areas where the road is hardly visible at all.

Our system has concentrated only on satellite imagery. We would like to investigate the interesting challenges that occur when satellite imagery is replaced with a more traditional line-drawn map, where perhaps an illustrative rendering approach might make more sense. Other challenges include methods for correcting the projection to compensate for tall vehicles consuming more of the projected space on the map, removing the existing vehicles on satellite imagery, and improving the background model to be more resilient to camera wobble.

10 Conclusion

We have presented a family of techniques for addressing the various challenges of projecting live traffic video onto a satellite image map. The three main focuses in this paper have been:

1. Assisting the user in specifying the inverse projection from the quasi-3D video space to the 2D map space;
2. Creating traffic visualisation for live imagery, a timeline plot and heatmap in real-time;
3. Devising a visual mapping scheme for the video data to density, change and uncertainty information for the timeline plot and heatmap.

We have provided a user-friendly and conceptually simple method of defining the projections using point correspondence and simple image editor-based tools for removing areas not to be projected; the combination of which we call *Flag and Cut*. Our rendering strategy uses a clustering method for inferring the colour of the road on both the video and the map area in order to perform histogram matching to blend the two roads together. The visual mapping strategy of our system does not attempt to automatically infer semantic detail from videos; instead it takes advantage of human visual intelligence to interpret the live scene through simple visualisations in order to gain more intuitive understanding of the video data.

Acknowledgments

This material is based upon work supported by the U.S. Department of Homeland Security's VACCINE Center under Award Number 2009-ST-061-CI0001. All Map data shown in this paper is ©Tele Atlas Imagery ©2011 Bluesky, Infoterra Ltd & COWI A/S, GeoEye, Infoterra Ltd & Bluesky, The GeoInformation Group, and used according to the fair use guidelines outlined in the Google Maps API documentation.

References

[1] G. Andrienko and N. Andrienko. A visual analytics approach to exploration of large amounts of movement data. In *Proceedings of the 10th international conference on Visual Information Systems: Web-Based Visual Information Search and Management*, VISUAL '08, pages 1–4, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] D. Ang, Y. Shen, and P. Duraisamy. Video analytics for multi-camera traffic surveillance. In *Proceedings of the Second International Workshop on Computational Transportation Science*, IWCTS '09, pages 25–30, New York, NY, USA, 2009. ACM.

[3] J. Arrospide, L. Salgado, M. Nieto, and R. Mohedano. Homography-based ground plane detection using a single on-board camera. *Intelligent Transport Systems, IET*, 4(2):149–160, Jun. 2010.

[4] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps (English Version)*. ESRI Press, 2010.

[5] R. P. Botchen, S. Bachthaler, F. Schick, M. Chen, G. Mori, D. Weiskopf, and T. Ertl. Action-based multifield video visualization. 14(4):885–899, 2008.

[6] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Press, 1999.

[7] M. Chen, R. P. Botchen, R. R. Hashim, D. Weiskopf, T. Ertl, and I. M. Thornton. Visual Signatures in Video Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1093–1100, 2006.

[8] S.-C. S. Cheung and C. Kamath. Robust background subtraction with foreground validation for urban traffic video. *EURASIP J. Appl. Signal Process.*, 2005:2330–2340, Jan. 2005.

[9] G. Daniel and M. Chen. Video visualization. *Proc. IEEE Visualization*, pages 409–416, October 2003.

[10] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. *Uncertainty in Artificial Intelligence*, pages 175–181, 1997.

[11] Y. He, H. Wang, and B. Zhang. Color-based road detection in urban traffic scenes. *Intelligent Transportation Systems, IEEE Transactions on*, 5(4):309–318, Dec 2004.

[12] M. Höferlin, B. Höferlin, D. Weiskopf, and G. Heidemann. Uncertainty-Aware Video Visual Analytics of Tracked Moving Objects. *Journal of Spatial Information Science (JOSIS)*, 2:in press, 2011.

[13] B. Hoummady. Method and device for managing road traffic using a video camera as data source. United States Patent No. US 6,366,219 B1, April 2002.

[14] C. Johnson and A. Sanderson. A next step: Visualizing errors and uncertainty. *Computer Graphics and Applications, IEEE*, 23(5):6 – 10, Sept–Oct 2003.

[15] M. Kilger. A shadow handler in a video-based real-time traffic monitoring system. In *Applications of Computer Vision, Proceedings, 1992., IEEE Workshop on*, pages 11–18, Nov 1992.

[16] Y. Kita. Background modeling by combining joint intensity histogram with time-sequential data. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 991–994, Aug. 2010.

[17] H. Kong, J.-Y. Audibert, and J. Ponce. General road detection from a single image. In *IEEE Transactions on Image Processing*, 2010.

[18] P. Kumar, S. Ranganath, H. Weimin, and K. Sengupta. Framework for real-time behavior interpretation from traffic video. *Intelligent Transportation Systems, IEEE Transactions on*, 6(1):43–53, Mar. 2005.

[19] S.-J. Lee and C.-S. Jeong. Real-time object segmentation based on gpu. In *Computational Intelligence and Security, 2006 International Conference on*, volume 1, pages 739–742, Nov. 2006.

[20] C.-T. Lu, A. P. Boedihardjo, and J. Zheng. Aitvs: Advanced interactive traffic visualization system. *International Conference on Data Engineering*, page 167, 2006.

[21] A. MacEachren. *How Maps Work*. The Guilford Press., 1995.

[22] P. McLauchlan, D. Beymer, B. Coifman, and J. Mali. A real-time computer vision system for measuring traffic parameters. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 495–503, Washington, DC, USA, 1997. IEEE Computer Society.

[23] M. Nekovee. Sensor networks on the road: the promises and challenges of vehicular adhoc networks and vehicular grids. *Proceedings of the Workshop on Ubiquitous Computing and e-Research*, 2005.

[24] A. Pang. Visualizing uncertainty in geo-spatial data. In *Workshop on the Intersections between Geospatial Information and Information Technology*, 2001.

[25] V. Pham, P. Vo, V. T. Hung, and L. H. Bac. *GPU implementation of Extended Gaussian mixture model for Background subtraction*, pages 1–4. IEEE, 2010.

[26] A. Robinson. *Elements of Cartography*. John Wiley & Sons, New York, 1953.

[27] M. Romero, J. Summet, J. Stasko, and G. Abowd. Viz-a-vis: Toward visualizing video through computer vision. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1261 –1268, Nov–Dec 2008.

[28] S. Shekhar, C. T. Lu, R. Liu, and C. Zhou. Cubeview: A system for traffic data visualization. In *Proc. IEEE Intelligent Transportation Systems*, pages 674–678, 2002.

[29] J.-C. Tai and K.-T. Song. Background segmentation and its application to traffic monitoring using modified histogram. In *Networking, Sensing and Control, 2004 IEEE International Conference on*, volume 1, pages 13–18, Mar. 2004.

[30] E. Tan and J. Chen. Vehicular traffic density estimation via statistical methods with automated state learning. *Advanced Video and Signal Based Surveillance, IEEE Conference on*, 0:164–169, 2007.

[31] L. Vibha, M. Venkatesha, P. G. Rao, S. N, P. D. Shenoy, K. R. Venugopal, and L. M. Patnaik. Moving vehicle identification using background registration technique for traffic surveillance. In *Proceedings of the International Association of Engineers International Multiconference of Engineers Computer Scientist*, Mar. 2008.

[32] Y. Wang, D. Krum, E. Coelho, and D. Bowman. Contextualized videos: Combining videos with environment models to support situational understanding. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1568 –1575, Nov–Dec 2007.

[33] J. Wood and J. Dykes. Spatially ordered treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14:1348–1355, 2008.

[34] J. Wood, A. Slingsby, and J. Dykes. Using treemaps for variable selection in spatio-temporal visualization.

[35] F. Zhu and L. Li. An optimized video-based traffic congestion monitoring system. In *Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining*, WKDD '10, pages 150–153, Washington, DC, USA, 2010. IEEE Computer Society.