2012 Ninth Conference on Computer and Robot Vision

3DTown: The Automatic Urban Awareness Project

Eduardo R. Corral-Soto*, Ron Tal*, Langyue Wang†, Ravi Persad†, Luo Chao†, Chan Solomon†,
Bob Hou*, Gunho Sohn† and James H. Elder*
*Dept. of Computer Science and Engineering / †Dept. of Earth and Space Science and Engineering
York University, Toronto, Canada
Email: {ecorral, rontal }@cse.yorku.ca, {wanglangyue, solomonchan.is }@gmail.com,
{ravi071, cluo, yhou, gsohn, jelder }@yorku.ca

***Abstract*—The 3DTown project is focused on the development of a distributed system for sensing, interpreting and visualizing the real-time dynamics of urban life within the 3D context of a city. At the heart of this technology lies a core of algorithms that automatically integrate 3D urban models with data from pan/tilt video cameras, environmental sensors and other real-time information sources. A key challenge is the three-dimensionalization of pedestrians and vehicles tracked in 2D camera video, which requires automatic real-time computation of camera pose relative to the 3D urban environment. In this paper we report preliminary results from a prototype system we call 3DTown, which is composed of discrete modules connected through pre-determined communication protocols. Currently, these modules consist of: 1) A 3D modeling module that allows for the efficient reconstruction of building models and integration with indoor architectural plans; 2) A GeoWeb server that indexes a 3D urban database to render perspective views of both outdoor and indoor environments from any requested vantage; 3) Sensor modules that receive and distribute real-time data; 4) Tracking modules that detect and track pedestrians and vehicles in urban spaces and access highways; 5) Camera pose modules that automatically estimate camera pose relative to the urban environment; 6) Three-dimensionalization modules that receive information from the GeoWeb server, tracking and camera pose modules in order to back-project image tracks to geolocate pedestrians and vehicles within the 3D model; 7) An animation module that represents geo-located dynamic agents as sprites; and 8) A web-based visualization module that allows a user to explore the resulting dynamic 3D visualization in a number of interesting ways.**

To demonstrate our system we have used a blend of automatic and semi-automatic methods to construct a rich and accurate 3D model of a university campus, including both outdoor and indoor detail. The demonstration allows web-based 3D visualization of recorded patterns of pedestrian and vehicle traffic on streets and highways, estimations of vehicle speed, and real-time (live) visualization of pedestrian traffic and temperature data at a particular test site. Having demonstrated the system for hundreds of people, we report our informal observations on the user reaction, potential application areas and on the main challenges that must be addressed to bring the system closer to deployment.

***Keywords*-mixed reality; visualization; distributed VR; tracking; camera calibration; rectification;**

I. Introduction

In recent years 3D Geographic Visualization Environments such as Google Earth and Microsoft Virtual Earth have demonstrated the potential power of making 3D geographic information broadly available over the internet. However, these tools only provide information about the static infrastructure of the urban environment. Since cities are by definition centres of activity, this limits the utility of these databases for many potential applications. On the other hand, the number of surveillance video cameras installed in urban areas grows every year as cameras become less expensive and as the demand for security and monitoring systems grows. This includes networks of cameras installed at universities, airports, train stations, shopping malls, etc.
With the potential benefit of having vast amounts of visual information comes the problem of viewing and analyzing the image data, which still today typically involves one or more human operators. This monitoring task can become overwhelming and inefficient if there are many cameras and few operators. For example if a pedestrian walks out of the field of view of camera A and enters the field of view of camera B, the operator may need some time to understand how the pedestrian is moving in the 3D world.

A. Prior Work

The problem of automatically mapping tracked agents in video to a 3D model has been studied previously in a number of different contexts. For example, Kanade et al ([1], [2]) developed a cooperative multi-sensor video surveillance and monitoring system (VSAM) for military applications that mapped the dynamics extracted from multiple distributed video cameras from different videos of a scene onto the appropriate areas of a common static 3D model of the corresponding scene, thus allowing the operator to see the projected scene dynamics in the context of the 3D model, thus obtaining a 3D situational awareness of the current dynamic environment. Some years later a similar system was reported by Sawhney et al [3] with the added feature that parts of the 3D model were texture-mapped with the projected live video images from the cameras. The system was implemented on commodity graphics hardware.
At roughly the same time, another group [4] began the development of a similar system called Augmented Virtual Environments (AVE) where GPS devices, orientation sensors and video cameras contained in a tracking backpack

that can be carried by a pedestrian were used to create augmented virtual environments. This system was later enhanced [5] with the introduction of background subtraction-based detection of moving objects followed by a pseudo-tracking step to extract track. More recently in [6] the authors divide the problem into four scenarios based on the number of cameras, overlapping of their fields of view, and types of motion: direct mapping (pedestrians), overlapping cameras with complex motion (sports), sparse cameras with simple motions (traffic), and sparse cameras with complex motion (clouds). In that work optical flow-based tracking and planar homographies are used to augment Aerial Earth Maps (AEM) with dynamic information from pedestrians and vehicles.

In practical video surveillance systems the camera pose is often changed by the operator. In this case it is necessary to update the camera matrix in order to re-compute the geolocation of tracked objects in the 3D model and to potentially perform automatic view updating of the model. None of the approaches discussed above focuses on this problem with the exception of [3] which describes a pose estimation method based on video image correspondences. However the method requires manual initialization and does not run in real-time.

In our work the goal is to develop a distributed system for sensing, interpreting and visualizing the real-time dynamics of urban life within the 3D context of a city. A clear advantage and main novelty in our system with respect to the approaches mentioned above is the introduction of an automatic on-line single-view method for updating the rotation matrix component of our virtual camera for the cases when the operator changes the pose.

Scenes contain several types of dynamic information. However we focus on typical, useful urban dynamic information, specifically walking pedestrians and moving vehicles captured by surveillance video cameras, and environmental signals coming from indoor temperature and motion sensors. In order to achieve our goal we have to integrate these different types of real-time dynamic information with a 3D virtual environment corresponding to an urban scene.

As a test site, we have selected a university campus that has a diverse mixture of building types, pedestrian and vehicle traffic. Over a 3D texture-mapped terrain surface model acquired from Google Earth, we have constructed both exterior and interior models of all of the campus buildings, using a diverse set of automatic, semi-automatic, and manual techniques (see below). Both recorded and live video data are acquired from pan-tilt-zoom (PTZ) video cameras situated at diverse sites over the campus. Tracking algorithms provide trajectories of pedestrians and vehicles in image coordinates. At a slower data rate, an automatic camera calibration algorithm computes the current pose of each camera relative to a world coordinate frame, allowing tracks to be back-projected to the 3D model. Intersections of these backprojections with the ground plane determine the geolocation of tracked agents, which are then represented as sprites,thus allowing the dynamics to be visualized in 3D within a web-based Google Earth environment. We also introduce an on-line system for monitoring temperatures in an indoor 3D building model which enables the temperature visualization of the whole floor of a building at once. We describe our methods to create photo-realistic indoor and outdoor 3D models of buildings, and how we augment Google Earth 3D urban models in real-time from surveillance cameras.At this point in time our work is mainly focused on detecting, tracking, geolocating and rendering agents within the field of view of a single camera rather than tracking agents between cameras.

The rest of the paper is structured as follows: Section II describes the overall system architecture, section III describes our virtual 3D world and how we create our own virtual building models, section IV describes how we map the PTZ camera 2D image points onto the virtual 3D world and how we automatically update the rotation matrix, section V explains how we track pedestrians and vehicles, sections VI and VII shows system integration details and results, section VIII includes our preliminary evaluation of the system, and section IX contains final comments plans for future work.

II. System Architecture Overview

Figure 1 shows the overall architecture of our system. The inputs to our system are surveillance video images, signals from environmental sensors, terrain surface 3D models provided by Google Earth, virtual 3D building models and geographic vector data such as roads and parking lots from Geographic Information Systems (GIS) and other map databases. The object tracking module uses a tracking algorithm to produce image coordinates of the tracked moving objects, and the camera projection matrix estimation module computes an updated camera matrix that is used to map these image coordinates onto Universal Transverse Mercator (UTM) coordinates in the 3D virtual world. Both camera pose information and the object UTM coordinates are written asynchronously onto XML (Extensible Markup Language) files on a webserver which are read by other modules in order to perform the rendering. The inputs from the Google Earth map server are virtual aerial 3D maps of the terrain. The inputs from the 3D building models module are the building models that we created, and the GIS/Road map data are additional inputs that we may use in the future. The rest of the blocks in the diagram are related to data storage and rendering. The details of all these modules will be explained in the later sections of the paper.

III. The virtual 3D world

The Google Earth Plug-in and its Javascript Application Programming Interface (API) provide a platform to develop various 3D applications by allowing users to incorporate

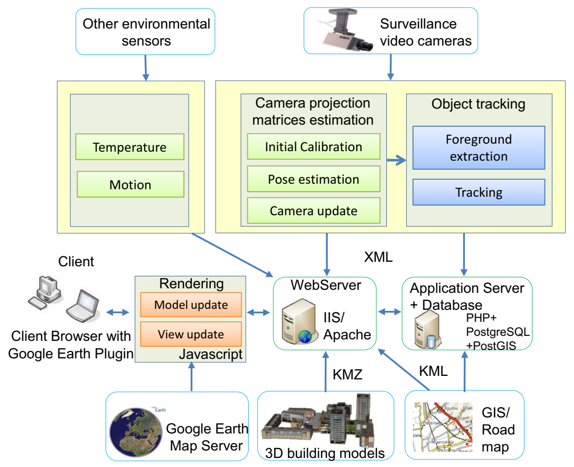

Figure 1. Simplified system architecture diagram.

their own 3D building models and other 3D data via KML files, thus enabling users to create customized 3D virtual worlds. The 3D photo-realistic building model is the most important component in the virtual world since buildings are the dominant man-made objects in urban environments. During the last two decades a number of algorithms and systems have been developed to construct 3D building models using various input data sources. With the development and wide application of Light Detection and Ranging (LIDAR) techniques, airborne and terrestrial laser scanning data has proven to be a valuable source of information for the construction of 3D building models. The 3D building models used in our system correspond to actual buildings from a university campus.

A. *Automatic Generation of 3D Photorealistic Prismatic Building Models*

For 3DTown, we commissioned airborne LIDAR scans of a university campus from an altitude of $2300m$ with a point distance density of about $1.9m$. Terrestrial LIDAR data was also collected with a mobile mapping system. We used these laser scanning data to reconstruct 3D photorealistic prismatic building models, using an established processing pipeline [7], consisting of: (1) point cloud registration, (2) automatic tie point collection, (3) geo-referencing, (4) surface modeling and (5) texture mapping. We used the Iterative Closest Point (ICP) algorithm [8] for the automatic registration of 3D point clouds from multiple scans. In order to ensure that the ICP solution converged correctly, accurate tie points were collected automatically by generating 2D images from 3D point clouds and performing automatic image matching. From these images, planar regions and their edges were extracted using a region growing method described in [9]. Those 2D edges were then transformed back to 3D to construct 3D planar surface [7]. For texture mapping a photogrammetric collinearity condition based method was used [10]. The optical images of the buildings used in the texture mapping were captured using a digital camera. We solve for the position and orientation of the camera via least squares and the collinearity condition. The tie points between the photos and LIDAR data were collected by the user through an interactive Graphical User Interface developed by our team. The textures are mapped to the surface model using a nearest neighbour resampling algorithm. Figure 5 shows an example of a reconstructed 3D building model.

IV. Coordinate Transformation between Cameras and the 3D Model: Determining Camera Matrices

A. *Back-projection of image points onto the 3D world*

In order to back-project tracked image locations $\vec{x}_i$ to corresponding 3D positions $\vec{X}_i$ in the world, we need to know both the internal and external parameters of the camera, captured by the projection matrix P: $\vec{x} = P\vec{X}$, which can be written as $P = [KR|\vec{t}]$, where K is a 3×3 calibration matrix containing the intrinsic parameters of the camera, R is a 3×3 rotation matrix and $\vec{t}$ is a translation vector.

We make the assumption that our cameras are installed at fixed an known locations, so that the translation vector t is known. However, computation of P is non-trivial since for PTZ cameras, both K and R can change on the fly.

In the current version of 3DTown, we make the simplifying assumption that the focal length is fixed. For offline camera data we select video segments where the zoom was not changed, and use an interactive calibration method [11] to estimate focal length and principal point, and make the assumption that skew is negligible and scaling parameters in x and y directions are identical, allowing complete identification of the internal camera matrix K. For the live camera, the camera focal length is fixed at its minimum value, and the camera matrix K is estimated using a standard offline method [12].

To perform the back-projection, the remaining step is to compute the camera rotation matrix R, which may change dynamically as the camera pans and tilts. We describe our solution to this problem in the next section.

B. *Estimating the Camera Pose via Manhattan Frames*

We use a novel, automatic, on-line, model-free method to maintain a continuous estimate of the rotation matrix R of each camera. Our method estimates the rotation independently for every frame by considering R as the product of two matrices

$$R = R_{M \rightarrow UTM} \cdot R_M, \tag{1}$$

Where R_M defines the rotation of the camera relative to the Manhattan frame (the canonical coordinate system defined by the orthogonal man-made structures in the scene) and $R_{M\to UTM}$ defines the orientation of man-made structures with respect to the UTM coordinate system. Since buildings are static, $R_{M\to UTM}$ need only be computed once. We now turn to the computation of R_M.

We can estimate the pose R_M of the camera with respect to the 3D scene structure by exploiting the rectilinear structure of typical urban environments. In particular, we exploit the so-called *Manhattan assumption* [13] , which states that an urban scene is generally dominated by planar surfaces that conform to three mutually orthogonal directions (e.g. vertical, streets and avenues). Under perspective projection, this regularity gives rise to convergence of the major lines of the urban environment onto a trio of vanishing points on the image plane. Coughlan and Yuille's algorithm [13] estimated the three principal Manhattan directions by modeling using a mixture model over the dense map of projected image gradients. Denis et al [11] later improved on these results using a sparser edge based formulation. Here we introduce a line-based method that we find provides further improvement in accuracy and reliability.

All three of these methods optimize the rotation Ψ between the camera and the Manhattan frame of reference by maximizing the likelihood function over a set of linear perspective cues $E = \{\vec{E}_1, \ldots, \vec{E}_N\}$:

$$\Psi^* = \arg\max_{\Psi} p(E|\Psi) \;=\; \prod_i p(\vec{E}_i|\Psi). \qquad (2)$$

The association of observations with the Manhattan directions is expressed through a mixture model:

$$p(\vec{E}_i|\Psi) \;=\; \sum_{m_i} p(\vec{E}_i|\Psi, m_i)p(m_i), \qquad (3)$$

where m_i is the 'Manhattan cause' of the line (vertical, horizontal(1), horizontal(2), background) and $p(m_i)$ is the prior over causes.

To compute the linear perspective cues E_i, we first detect and localize image edges to sub-pixel accuracy [14]. These edges are then grouped into lines using a Hough transform technique [15] that uses a kernel-based voting scheme to propagate the uncertainty of edge observations onto the parameter domain. A common problem with Hough methods is the multiple response problem: multiple peaks detected in the Hough map that correspond to a single linear structure in the scene. In our approach, these multiple responses are avoided by employing a stepwise probabilistic subtraction method [16] that subtracts off the contributions of edge observations that correspond to previously detected lines.

The resulting set of detected image lines can be used to recover R_M by considering the *Gauss sphere* representation of the problem [17]. A detected line in the image plane, together with the optical centre of the camera define an *interpretation plane*. The space formed by the normal vectors of all possible interpretation planes is called the *Gauss*

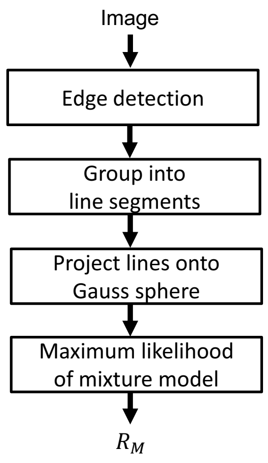
Image
Edge detection
Group into line segments
Project lines onto Gauss sphere
Maximum likelihood of mixture model
R_M

Figure 2. Estimating the camera rotation matrix.

sphere. Under perspective projection, the interpretation plane normals of parallel lines in the 3D scene are coplanar, and the normal vector to this plane lies in the direction of these parallel lines. Thus a good choice for the observable E_i used in the likelihood $p(\vec{E}_i|\Psi, m_i)$ is the angular error formed between the interpretation plane of an observed line and the putative 3D orientation vector of the corresponding Manhattan direction.

We learn the values of the priors $p(m_i)$ and the parameters of the distribution of the error functions using a public ground-truth database [11], and optimize the likelihood over the unknonwn Euler angles Ψ using a multi-start version of a BFGS gradient-descent algorithm [18]. Figure 2 summarizes the complete pipeline for camera pose estimation. The output of this algorithm is the rotation matrix that defines the pose of the camera with respect to the Manhattan frame, and the inverse R_M of this matrix thus defines the transformation from the camera coordinate system to the Manhattan coordinate system. An example result from the algorithm is shown in Figure 3. Estimation of camera pose takes roughly 8 seconds on a standard desktop computer. In practice we find this is sufficiently fast for typical surveillance cameras, where cameras tend to stay in the same pose for minutes or even hours at a time. In order to improve the robustness of the system to noise and illumination changes we perform time averaging of the computed rotations [19] over a window of 5 frames.

V. Scene Dynamics: Tracking Pedestrians and Vehicles

Detection and tracking of pedestrians and vehicles is based upon a probabilistic background subtraction computation [20], [21]. We model the colour of each image pixel as a mixture of two multivariate normals $\eta(\vec{\mu}_i, \Sigma_i)$, $i \in \{1, 2\}$ corresponding to background and foreground processes. To reduce the effects of shadows and illumination variation, we

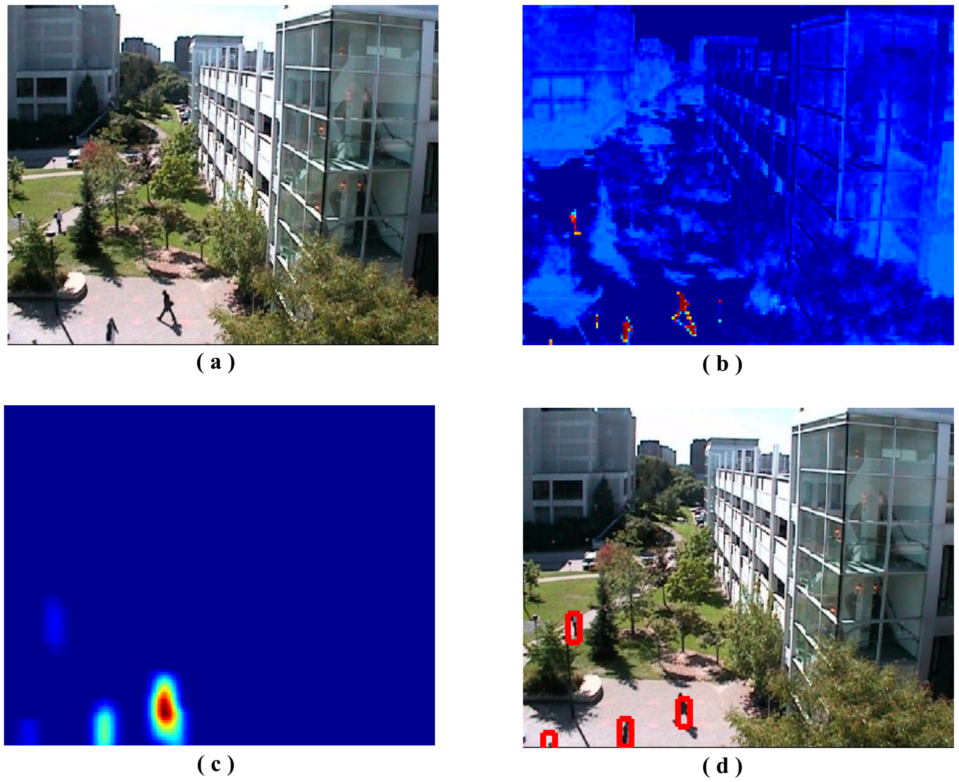
Figure 4. The stages of our tracking algorithm. a).-Input image, b).-Current foreground posterior map, c).-Thresholded, smoothed, normalized absolute difference of current and previous posterior maps, d).-Tracking boxes overlayed on the input image.

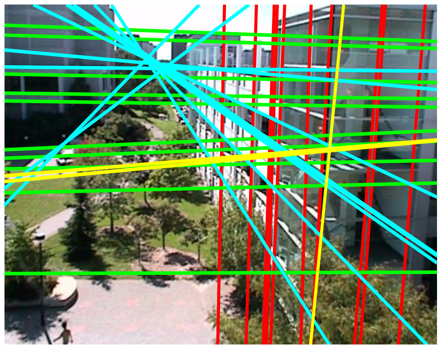
Figure 3. Example of Manhattan frame estimation. The cyan and green lines correspond to the two horizontal axes, red lines correspond to the vertical axis, and yellow lines are "non-Manhattan" lines that do not conform to any of the three principal directions.

factor out intensity and work in the residual two-dimensional colour subspace $\vec{X}$ [21].

Maximum likelihood estimates of the mean $\vec{\mu}_i$, covariance matrix Σ_i and mixing coefficient ω_i of both mixture components are updated on each frame using an incremental version of the EM algorithm [20]. Note that the mixing coefficients ω_i for a specific pixel represent the prior probability of foreground and background at that pixel. We assume that on average, over time, each pixel corresponds more frequently to background than foreground, and thus associate the larger mixing coefficient with the background process. Representing background and foreground hypotheses as H and $\bar{H}$ respectively, the prior probabilities are thus given by

$$p(H) = max\left(\omega_1, \omega_2\right), p(\bar{H}) = min\left(\omega_1, \omega_2\right). \quad (4)$$

For each pixel of every incoming image, we compute the posterior probability of the hypothesis H that a pixel belongs to the foreground given its colour $\vec{X}$:

$$P(H|\vec{X}) = \frac{P(\vec{X}|H)P(H)}{P(\vec{X}|H)P(H) + P(\vec{X}|\bar{H})P(\bar{H})}. \quad (5)$$

In order to track moving objects of interest while ignoring environmental motion (e.g. slow-moving trees), we have developed a simple but efficient and effective tracking algorithm. Let P^n and P^{n-1} represent the foreground posterior probabilities corresponding to the current and previous frames. Then the algorithm consists of 5 computational steps:

1) We compute the absolute two-frame difference $P_d^n = |P^n - P^{n-1}|$.
2) We zero out differences P_d^n below a threshold T

3) We apply a 2D Gaussian low-pass filter to P_d^n. This produces smooth blobs that can be tracked relatively easily (Fig.4c).
4) We extract the set of image locations $S^n = \{\vec{\mu}_1, \ldots, \vec{\mu}_{K^n}\}$ corresponding to the maxima of each Gaussian-like blob, where K^n is the number of objects detected in the current image. Note that K^n (and S^n) can change from image to image depending upon the objects that enter or exit the camera's field of view. Our experiments have shown that S^n can be computed effectively using 5-10 iterations of the Mean-Shift algorithm [22], however in practice we find that a simple peak detection algorithm produces very similar results in a fraction of the time.
5) The actual tracking of the ith object $\vec{\mu}_i \in S^n$ at time n is performed by finding the nearest neighbour $\vec{\mu}_{j^*} \in S^{(n-1)}$ at time $n-1$, which is done by finding its index
$$j^* = \arg\min_j \parallel \vec{\mu}_i - \vec{\mu}_j \parallel, \forall j \in \{1, \ldots, K^{n-1}\}, \quad (6)$$
from which we compute motion vectors $\vec{v}_i = \vec{\mu}_i - \vec{\mu}_{j^*}$ that can be used for temporal smoothing of the tracks, and for speed estimation of the moving objects.

Figure 4d shows an example frame of the algorithm tracking four pedestrians.

VI. Monitoring of Environmental Signals in Indoor Models

One potential application of 3DTown technology is for the monitoring and analysis of energy usage and how that relates to the flow of people through the city. As an initial step toward this application, we have integrated within one of the buildings in our 3DTown demonstration a distributed sensing network that monitor the ambient air temperature. To visualize this information, we colour-code rooms of the building according to their temperature, with blues representing cooler temperatures (from 16^oC), and oranges and reds representing warmer temperatures (to 30^oC) - see Fig. 6.

VII. Data Integration and Results

We have developed an intuitive web-based graphical user interface that allows the user to select an available real-time surveillance video camera located in the 3D model, activate the 3D visualization of tracked pedestrians and vehicles, change the 3D view of the scene, and query indoor temperature. Estimated 3D pedestrian and vehicle coordinates are stored in an XML file, which is read by the rendering program. Tracked pedestrians are rendered as simple 2D sprites and vehicles and buses as simple 3D models in DAE (Digital Asset Exchange) format that can be loaded and rendered directly in Google Earth. Figures 5 and 6 show examples of visualizations provided by our web-based user interface.

VIII. Preliminary Evaluation

Prior systems for 3D dynamic visualization of urban scenes [2], [3], [4], [6] have generally not been systematically and quantitatively evaluated, and there is no standard method for such an evaluation at this point. Ultimately, we intend to conduct a human-in-the-loop usability study within the context of a specific set of tasks. At this point, however, we can report some specific quantitative performance parameters for specific modules of our system, as well as qualitative observations gleaned from demonstrations to hundreds of observers.

The accuracy of our 3D building models is on the order of $5cm$. The average error of our automatic camera pose estimation module, measured on a standard public dataset (YorkUrbanDB) is 2.5 deg, which compares favourably with other published single-frame approaches [11].

While our tracking module operates at 8 frames per second on a standard PC, our camera pose algorithm takes about 8 seconds to estimate the camera pose from a single frame: thus it is useful for intermittent pan/tilt operation, but not for continuous smooth pursuit.

We have received substantial qualitative feedback on the system as a whole from over one hundred observers. The most frequent observations are:

1) The ability to maintain the 3D dynamic model through pan/tilt shifts of the camera is seen as a very powerful and important capability.
2) Integration of outdoor with indoor models and environmental sensing is perceived as novel, interesting and useful.
3) Delays are sometimes introduced when retrieving model information from Google Earth, particularly when camera pose changes, and these delays are seen as objectionable. This suggests a different geoserver framework that is optimized for dynamic visualization.
4) The sprites currently in use are too simplistic and should be upgraded to fully articulated avatars.
5) Automatic labeling of actions (e.g., walking, running...) is perceived as a valuable feature for future versions of the system.
6) With only a single live camera, avatars appear and disappear when they enter and leave the field of view, and this is seen as objectionable. More convincing live demonstration will depend upon a facility with multiple, ideally overlapping fields of view.

IX. Conclusions and Future Work

In this paper we have reported preliminary results from our prototype 3DTown distributed system which allows real-time 3D visualization and analysis of scene dynamics for

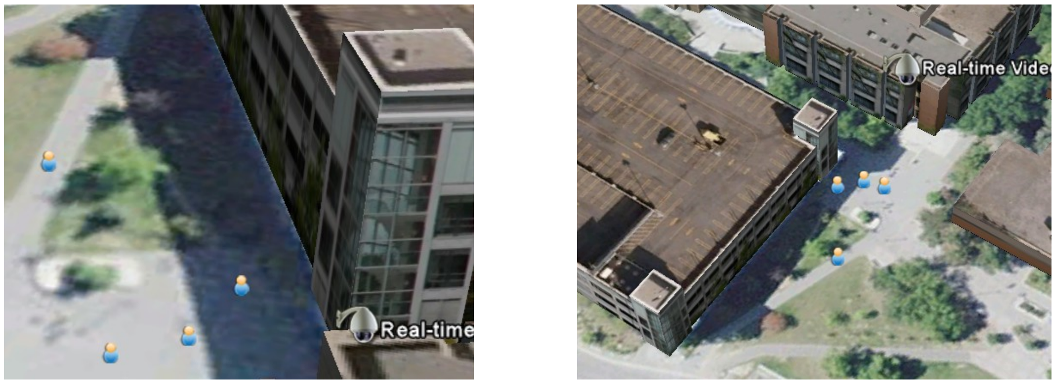

Figure 5. Two different views of the 3D visualization of the four tracked pedestrians from Fig. 4-d .

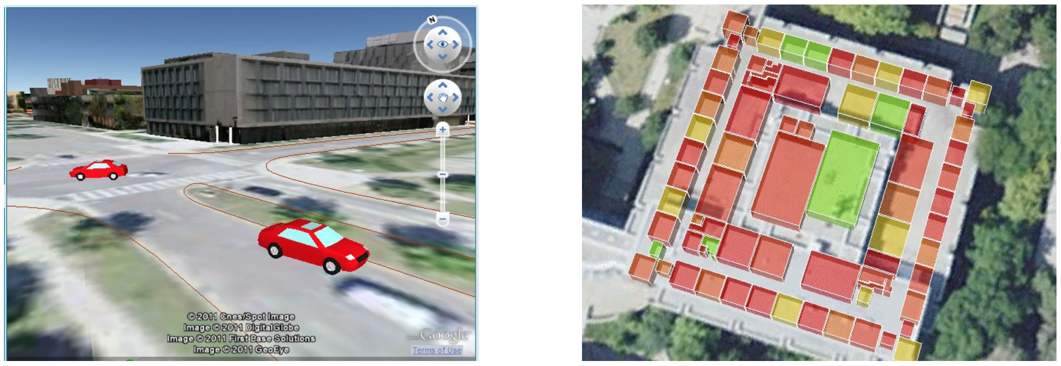

Figure 6. Left: An example of vehicles tracking in the 3D virtual world. Right: Example of room temperatures visualization.

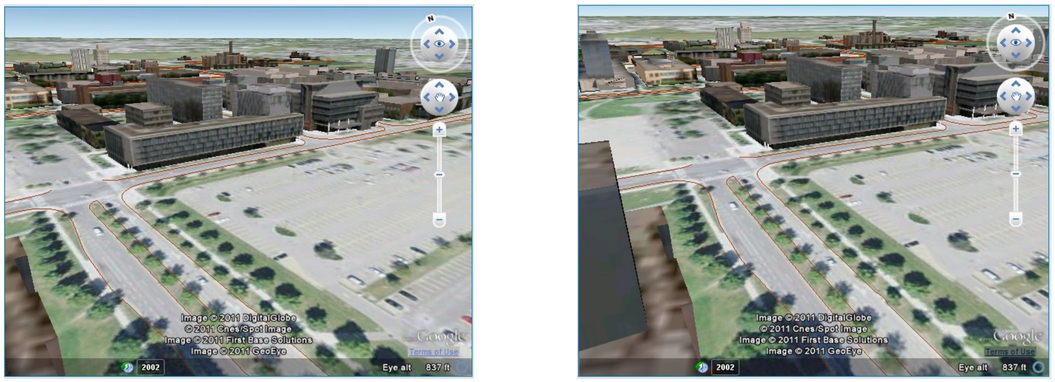

Figure 7. Example of camera pose update. Left: View before video camera pose change. Right: Automatically-updated view after the video camera pose change.

urban environments, within a flexible distributed architecture. To demonstrate the system, we created a 3D model of a university campus, integrated with Google Earth terrain data. Three-dimensionalization of data extracted from 2D video cameras is achieved by an algorithm that uses the Manhattan structure of the urban scene to automatically estimate the camera pose. This allows automatically tracked pedestrians and vehicles to be geo-located and thus represented as sprites in the 3D model. Our web-based interface allows the user to browse the 3D model and visualize the scene dynamics of a particular site in real-time while changing the view of the 3D visualization. If the pose of the video camera changes, our system will automatically update the corresponding projection matrix to maintain accurate geolocation of the scene dynamics.

Demonstrations of the 3DTown system for hundreds of people have yielded substantial feedback that has been helpful in planning future work, which will include 1) Expanding the system to include more cameras, thus providing complete coverage of an entire building or thoroughfare, 2) Improvements to our tracker, 3) Minimization of rendering delay by migrating the system to an OpenGL platform, currently under development, 4) Introduction of full-articulated avatars, 5) Incorporation of automatic action labelling, 6) Introduction of a standard method for evaluating the efficacy of the system as whole.

REFERENCES

[1] T. Kanade, R. Collins, A. Lipton, P. Anandan, and P. Burt, "Cooperative Multisensor Video Surveillance," *Proc. of DARPA Image Understanding Workshop*, pp. 3–10, 1997.

[2] T. Kanade, R. Collins, A. Lipton, P. Burt, and L. Wixson, "Advances in Cooperative Multi-Sensor Video Surveillance," *Proc. of DARPA Image Understanding Workshop*, pp. 3–24, 1998.

[3] H. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna, "Video Fashlights - Real Time Rendering of Multiple Videos for Immersive Model Visualization," *Thirteenth Eurographics Workshop on Rendering (2002)*, pp. 157–168, 2002.

[4] U. Neumann, S. You, J. Hu, B. Jiang, and J. Lee, "Augmented Virtual Environments (AVE): Dynamic Fusion of Imagery and 3D Models," *VR03, March 2003*, 2003.

[5] I. Sebe, J. Hu, S. You, and U. Neumann, "3D Video Surveillance with Augmented Virtual Environments," *IWVS03, November 7, 2003, Berkeley, California, USA*, pp. 107–112, 2003.

[6] K. Kim, S. Oh, J. Lee, and I. Essa, "Augmenting Aerial Earth Maps with Dynamic Information," *IEEE International Symposium on Mixed and Augmented Reality 2009, Science and Technology Proceedings*, pp. 19–22, 2009.

[7] J. Li-Chee-Ming, D. Gumerov, T. Ciobanu, and C. Armenakis, "Generation of Three-Dimensional Photo-Realistic Models from LIDAR and Image Data," *Proceedings 2009 IEEE Toronto International Conference - Science and Technology for Humanity*, pp. 445–450, 2009.

[8] P. Besl and N. McKay, "A Method for Registration of 3-D Shapes," *IEEE Trans. Pat. Anal. and Mach. Intel.*, vol. 14, no. 2, pp. 239–256, 1992.

[9] D.G.Bailey, "Raster Based Region Growing," *in Proceedings 6th New Zealand Image Processing Workshop, Lower Hutt, New. Zealand*, pp. 21–26, 1991.

[10] F. H. Moffit and E. M. Mikhail, *Photogrammetry*. Harper & Row, Inc., 1980.

[11] P. Denis, J. Elder, and F. Estrada, "Efficient Edge-Based Methods for Estimating Manhattan Frames in Urban Imagery," *European Conference on Computer Vision*, pp. 197–210, 2008.

[12] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[13] J. Coughlan and A. Yuille, "Manhattan World: Compass Direction from a Single Image by Bayesian Inference," *International Conference on Computer Vision.*, vol. 2, pp. 941–947, 1999.

[14] J. H. Elder and S. W. Zucker, "Local Scale Control for Edge Detection and Blur Estimation," *Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 7, pp. 699–716, 1999.

[15] R. Duda and P. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, vol. 1, no. 15, pp. 11–15, 1972.

[16] O. Barinova, V. Lempitsky, and P. Kohli, "On the Detection of Multiple Object Instances using Hough Transforms," *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[17] S. T. Barnard, "Interpreting perspective images," *Artificial Intelligence*, vol. 21, no. 4, pp. 435–462, 1983.

[18] M. Avriel, *Nonlinear Programming: Analysis and Methods*. Prentice Hall, 1976.

[19] C. Gramkow, "On averaging rotations," *Int. J. Comput. Vision*, vol. 42, no. 1-2, pp. 7–16.

[20] N. Friedman and S. Russel, "Image Segmentation in Video Sequences: A Probabilistic Approach," *In Proc. UAI*, pp. 175–181, 1997.

[21] J. Elder, S. Prince, Y. Hou, M. Sizintsev, and E. Olevskiy, "Pre-Attentive and Attentive Detection of Humans in Wide-Field Scenes," *International Journal of Computer Vision*, vol. 72, no. 1, pp. 47–66, 2007.

[22] C. Dorin and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, 2002.