Robust and Efficient Change Detection Algorithm based on 3D Line Segments

Tom Zohar
Signal and Image Processing Lab.
Dept. of Electrical Engineering
Technion, Israel Institute of Technology
Haifa 32000, Israel
tomzohar@gmail.com

Ido Ariav
Signal and Image Processing Lab.
Dept. of Electrical Engineering
Technion, Israel Institute of Technology
Haifa 32000, Israel
idoariav@gmail.com

Meir Bar-Zohar
Signal and Image Processing Lab.
Dept. of Electrical Engineering
Technion, Israel Institute of Technology
Haifa 32000, Israel
meirb@visionsense.com

***Abstract*—In this paper we examine and improve a new approach for change detection (introduced in [1]) which is based on the appearance and disappearance of 3D line segments as seen in a new image. These 3D line segments are estimated from a set of learning images taken from arbitrary viewpoints and under arbitrary light conditions in an unsupervised manner.**

The main advantage of the proposed method lies in the fact that the change detection is performed by comparing line segments, and not surfaces or gray levels. Computing 3D surfaces in an image can be computationally intensive, and other methods such as image subtraction or cross-correlation are sensitive to lighting conditions and changes in viewpoints. Moreover, most man-made objects such as buildings, cars, and even cities viewed from above consist mainly of straight lines, and therefore this method is highly applicable for such structures.

The proposed algorithm first focuses on the reconstruction of a set of 3D line segments forming a certain 3D scene using a set of 2D line segments obtained from the learning images in an unsupervised manner, without any prior knowledge on the cameras' positions or relative distance. In the change detection stage, we use the reconstructed 3D scene of line segments to check if changes, such as appearance or disappearance of objects, have occurred in a given test image. This test image can be taken from arbitrary viewpoint and under arbitrary lighting conditions. Our change detection algorithm not only distinguishes between the states of "*changed*" and "*not-changed*" line segments, it also classifies the "*changed*" line segments as appeared - objects that entered the scene in the test image, and disappeared - objects that left the 3D scene reconstructed from the lines of the learning images.

I. Introduction

In most change detection problems, the goal is to successfully reconstruct a 3D scene using N given learning images, and then decide if a significant change has taken place based on a $N+1$ test image. The common approach to such problems is to estimate the 3D model of the scene using the associated BRDF[1] and then make a decision based on the knowledge we have on the viewing position and lighting conditions of the $N+1$ image. If there is a significant change in the $N+1$ image compared to the estimated model we decide that a change has occurred.

The estimation of the 3D model is usually done in an unsupervised manner from the N learning images. Note that in this new approach, we do not require the learning images to be taken from the same viewpoint and in fact they can be taken from arbitrary viewpoints. A good example for this can be pictures of an urban scene taken by a passing satellite. Each image can be taken from an arbitrary different location and viewpoint. Another example is of cars in a parking lot captured by different surveillance cameras. The majority of examples in this paper will come from the latter scenario. It is important to emphasize that we deliberately do not restrict ourselves to the case of buildings, which have long and relatively easy to detect straight lines, but rather focus on the case of cars, which mainly consist of short lines or curves.

In this paper we examine, generalize and improve a new method (introduced in [1]) for reconstructing the 3D model and detecting changes – estimating the scene model based only on straight line segments. Straight lines appear in almost any man-made objects and therefore this method is applicable to a variety of scenarios – buildings, cars, urban areas, etc. The major advantage to this approach is that working with line segments is computationally less complicated than working with the full set of pixels and also less sensitive to changes in lighting conditions and viewpoints. Moreover, most curves can be decomposed into short straight lines and therefore this method will be applicable to these cases as well.

II. Related Work

Most earlier work on change detection methods was performed on images taken from a stationary camera at a known position. This camera produced a sequence of images of the same scene from the same viewpoint and the change detection algorithm determined if a change has occurred based on these images. In those algorithms a 3D model of the scene was not estimated and the change detection was based on pixel value (intensity) differences between different images in the sequence. The main drawback of such methods is their strong sensitivity to illumination and noise. Prominent examples for such methods are image differencing and background modeling methods.

Later on, several methods for dealing with the cases of non-stationary image sequences arose. A great deal of work has been done in the field of moving object detection in video sequences. These methods require the images to be taken with short time gaps between images and small changes in viewpoint. Therefore, they are not compatible to deal with cases in which the distance between viewpoints is large and the time between images is long (hours\days). There also exist several methods that are based on the reconstruction of 3D surfaces. However, these methods are computationally

[1] Bidirectional Reflectance Distribution Function

intensive and are sensitive to changes in illumination. They are also known to perform poorly around object boundaries.

Finally, several change detection algorithm make use of straight lines in order to detect changes. Such are [2] that make use of 2D line segments in their algorithm, but their method is specifically designed for aerial images where the images can be registered using an affine transformation. Li et al [3] provided a method of detecting urban changes from a pair of satellite images by identifying changed line segments over time. Their method does not estimate the 3D geometry associated with the line segments and takes a pair of satellite (aerial) images as input where line matching can be done by estimating the homography between the two images.

Eden and Cooper [1] proposed a solution for the change detection problem using 3D reconstructed scene based on line segments. We generalize their solution for the uncalibrated set of cameras case, without making any prior assumptions on the cameras' positions or time between images, and improve performance and robustness by adding a K-Nearest Neighbors (KNN) algorithm to the change detection decision stage.

III. Reconstructing the 3D Scene from Learning Images

A. Extracting 2D Line Segments in Learning Images

In order to reconstruct a 3D scene of 3D line segments we first need to match 2D line segments across all learning images. This procedure imposes two difficulties that are well known in literature - efficient extraction of 2D lines in an image and efficient matching of 2D lines across images.

When an algorithm for extracting 2D lines such as Hough transform is applied on an image, the lines are often fragmented into small segments that diverge from the original line segments. In this case, instead of getting one 2D line segment representing a real straight line, we get several line segments that each represent a part of that one straight line. This situation causes many problems when later we try to match each segment to a segment in other images. In order to try and overcome this difficulty, we divide the learning image into a number of equally sized blocks before applying edge detection and line extraction. The idea is that now both the edge detection algorithm and the line extraction algorithm will work with local thresholds instead of global thresholds in the entire image. In this way, blocks that are either rich in details (and straight lines) or very smooth (almost no lines) will benefit from that change. At the end of the procedure the line segments from all the blocks are merged into a single array of lines.

B. Matching 2D Line Segments across Images

Once an array of lines is extracted from each learning image, a method for efficient and reliable matching is needed. Matching of line segments across images is known to be a difficult task due to its exponential complexity in image number. Therefore, our method uses the geometric constraints of epipolar geometry in order to decrease that complexity and to get better matching results.

We estimate the Fundamental matrices, $F_{i,j}$, in a supervised manner for each pair of images I_i and I_j, where $i, j \in \{1, \dots, N\}$. This is done by manually selecting and matching points in the images, using the camera's internal parameters for calibration and using the Gold Standard method as described in [4]. The projective camera matrix P_j for image I_j is calculated from a Fundamental matrix $F_{i,j}$, for all $j \in \{1, \dots, N\}$.

By using the epipolar line constraint on each endpoint of a line segment in a certain image, we can eliminate all the 2D lines in the other images that are not a good candidate for matching and therefore produce optional good candidates for matching. For each line in image I_1, we calculate the two epipolar lines in image I_j corresponding to the two line segment endpoints using $F_{1,j}$, and we search for all line segments that lie inside the area between the two epipolar lines, in image I_j, where $j \in \{2, \dots, N\}$. Since the 2D line extraction algorithm sometimes yields imperfect lines, and thus endpoints, we allow some freedom, and lines with endpoints that lie several pixels away from the epipolar lines are also considered good candidates.

Then, for each line in image I_1, a match score is calculated for all good candidates in image I_j and the corresponding line in image I_j will be that with the highest match score.

Another issue that needs to be considered while matching 2D line segments is the fact that methods of measuring correspondence between two line segments that differ in angle as a result of different point of view in different images, will give problematic results. Therefore, in order to overcome this issue, before any attempt to calculate a match score for two line segments, both images are rectified so that the compared lines are parallel.

IV. 3D Reconstruction and Assembling Wire-Frame Models

In order to reconstruct the 3D scene of 3D line segments represented in the learning images, our method uses only 2D line segments that have matches across at least M learning images, where $M \leq N$. This way, many of the 2D line segments that were not extracted correctly (too short, fragmented etc.) are eliminated and do not take part in the 3D reconstruction. This helps improve the accuracy and reliability of the reconstructed 3D scene.

The reconstruction procedure of 3D lines is done by estimating the two points in space, for each 3D line, that corresponds to the line's endpoints in M images. Let $J \subseteq \{1, \dots, N\}$ be the subset of image indices in which the 3D line endpoint X was projected to and found. Let the point x_j be the projected 2D endpoint in image I_j, where $j \in J$. Due to line extraction errors, the rays back-projected from the points x_j are skew. This means that there will not be a point X which exactly satisfies $x_j = P_j X$, where P_j is the projective camera matrix for image I_j and $j \in J$, so a least squares solution is estimated.

Once a set of 3D lines, represented by their endpoints, was obtained from the linear reconstruction algorithm, a non-linear algorithm is applied in order to minimize the Euclidian distance of the projected lines to the original 2D lines in all views. Here we use the assumption that for each 3D line segment, a corresponding set of 2D line segments is available in all views. Each 3D line segment is represented by two normalized 3D endpoints, therefore represented by six parameters in 3D space.

In order to solve this minimization problem we use the Nelder-Mead method and the following cost function as used in [1]

$$L_{new} = \text{argmin}_{L \in \{R^3, R^3\}} [\sum_{i=1}^{n} d_l(l_i, l_i') + \beta \sum_{i=1}^{n} d_s(l_i, l_i'')] \quad (1)$$

where L is the linearly reconstructed 3D line, L_{new} is the improved 3D line segment of L, l_i is the corresponding 2D line segment in image I_i, l_i' is the projection of L to image I_i as an infinite line, and l_i'' is the projection of L to image I_i as a finite line segment.

$d_l(l_i, l_i')$ is the distance metric between an infinite line and a line segment and is computed as seen in the following formula

$$d_l(l, l') = \sqrt{\frac{1}{|l|} \sum_{p \in l} d_p^2(p, l')} \quad (2)$$

where d_p is the perpendicular distance of a point p to an infinite 2D line. The line segment l is divided to points p and an average of the point to line distances is calculated.

$d_s(l_i, l_i'')$ is the distance metric between two line segments. It is computed in the following manner

$$d_s(l, l'') = \sqrt{\frac{1}{|l|} \sum_{p \in l} d_{ps}^2(p, l'')} + \sqrt{\frac{1}{|l''|} \sum_{p'' \in l''} d_{ps}^2(p'', l)} \quad (3)$$

where d_{ps} is the minimum distance between a point and a line segment. Both line segments l and l'' are divided into points p and p'' and an average of the point-to-line-segment distances is calculated for both lines.

β appearing in Eq. (1) is used to control the convergence of the local search algorithm. β is selected to be a positive number close to zero ($0 < \beta \ll 1$). In this way, at first the algorithm tries to converge to the correct infinite line and after that the second part in the cost function (succeeded by β) becomes more dominate and search for the optimal endpoints for the 3D line segment. Example of the improvement of the non-linear algorithm is shown in Figure 1.

In the case of a scene of cars in a parking lot, after the 3D reconstruction, the models of the cars were assembled from individual lines and there were still some errors due to degeneracies of some lines due to their 3D orientation with respect to the cameras' viewpoint. This usually occurs when a 3D line segment and the camera center lie on the same plane. In addition, since a certain line in one picture can have a different length from its correspondent line in some other image due to line extraction imperfections, additional constraints are added in order to overcome those errors.

We use the geometrical properties of the cars, having obvious wire frame outlines, in order to add these constraints. Line endpoints that qualify a distance criterion to a different line endpoint in 3D, as well as a distance criterion in all three learning images, can be assumed to be originated from two attached lines in the original scene, having a mutual endpoint. We use a pair-wise checking algorithm, and define two radios thresholds – for 3D space and for 2D space for all line segment endpoints. Since small errors in the 2D line segment extractions can lead to large error in 3D space, we choose a larger threshold for 3D space with respect to the scene, than the threshold for 2D space with respect to the scene. We do not use any prior information, assumptions or model for the structure of the cars, but solely depend on the described criterions. Using an iterative algorithm, endpoints are joined together until there are no such points that satisfy the criterions.

Each wire-frame model is formed as an undirected graph $G = (V, E)$ where the set of edges represent 3D line segments and the vertices represent their endpoints. Instead of minimizing the objective function for each line segment separately, here we minimize an objective function for all line edges in the wire-frame graph model. The cost function for wire-frame minimization problem is the following formula also used in [1]

$$G^* = argmin_{V \in R^{3N_V}} \sum_{e \in E} \left(\sum_{i=1}^{n} d_l\left(l_{i_e}, l'_{i_e}\right) + \beta \sum_{i=1}^{n} d_s\left(l_{i_e}, l''_{i_e}\right) \right) \quad (4)$$

where N_V is the number of vertices in a wire-frame model and l_{i_e} is the line in image I_i associated with edge $e \in E$ in graph $G = (V, E)$. d_l,d_s are as defined in Eq. (2) and Eq. (3).

As in the non-linear reconstruction, we also use here the Nelder-Mead optimization algorithm to find the vertices that minimize the cost function, constraining together sets of line segments and solving this optimization algorithm for every set that hold the closeness constraints.

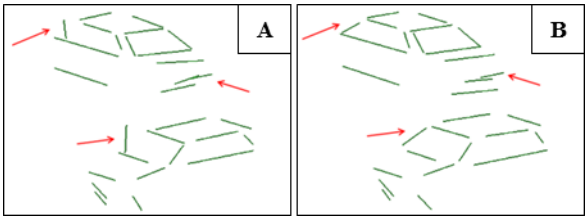

Fig. 1. *(A) The 3D reconstruction of two cars after using linear reconstruction only. (B) The same reconstruction after applying the non-linear algorithm. Notice the improvement of the reconstructed lines marked by a red arrow.*

V. Change Detection

A. The Concept of the Proposed Algorithm

Our change detection algorithm is based on the appearance and disappearance of line segments throughout an image sequence. In our change detection problem, given a new test image the algorithm must decide whether a significant new object has appeared in the region or whether an object in the region has left based on the 3D scene reconstructed from the line segments from the learning images as in section III. The output of this algorithm is a visual one that assists the viewer by marking the 2D lines in the test image in different colors according to changed or unchanged state.

The test image can be taken from an arbitrary viewpoint, different from all viewpoints of the learning images. We estimate the 3D scene which consists only of long and short straight lines, since estimating the complete 3D surface under varying illumination conditions and in the existence of specular highlights is often impractical. For man-made objects and for general 3D curves, straight line approximations are usually appropriate and effective. Our method detects changes by interpreting reconstructed 3D line segments and 2D line segments detected in learning and test images.

Our change detection method is composed of two procedures that eventually assign a state to each 2D line extracted in the test image (the $N + 1$ image) and for each 3D line in the reconstructed scene. The possible states for the 2D lines are "*not-changed*" or "*changed*", when "*changed*" means that this line does not appear in the 3D scene – therefore it's a new line. The possible states for the 3D lines are also "*not-changed*" and "*changed*", when this time "*changed*" means that the 3D line does not appear in the new image and therefore the line belongs to an object that left the scene.

B. Change Detection Tests for 2D Line Segments

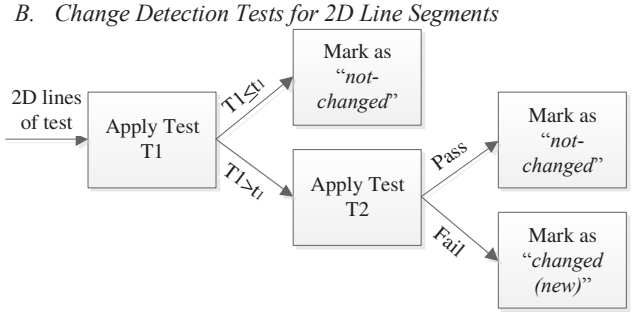

Fig. 2. *Tests T1 and T2 applied to determine the state of each 2D line in the new test image*

Once a new test image is given, all 2D line segments in that image are extracted in the same manner as described in section III.A. In order to determine the state of each 2D line extracted in the test image, we use two statistical test T1 and T2 and the procedure that appears in Figure 2 and used in [1].

First we apply test T1 for every 2D line segment extracted from the test image. This test is designed to check how well a 2D line in the test image fits the reconstructed 3D scene. If for a specific 2D line in the test image exists a close enough projected 2D line from the 3D scene then this line is not new – it existed also in the learning images or else it wouldn't have appeared in the 3D reconstruction. If, however, such close enough projection does not exist then this means the line does not appear in the 3D scene and therefore it could belong to a new object that appeared in the test image.

In order to find the closest projected 2D line we used Eq. (3) where here l_i is the 2D line segment in the test image and l_i'' is the projected 2D line segment. After calculating $d_s\,(l_i, l_i'')$ for all projected lines, if the distance of the closest projection is less than a t_1 threshold, the algorithm marks the state of this 2D line as "*not-changed*". If the distance is greater than a t_1 threshold, we apply T2 test for that line.

A demonstration of test T1 can be seen in Figure 3. In this example the lines that were marked as "*not-changed*" (red lines) are indeed only the ones that are relatively close to the projections from the 3D scene (green lines). The rest of the 2D lines are marked as "*changed*" (blue lines) and indeed we can see that they do not appear in the reconstructed 3D scene of line segments (not close to any of the green lines).

After applying test T1 for every 2D line extracted in the test image we turn to apply test T2 if necessary. T2 test is applied to all the 2D lines in the test image that were not marked as "*not-changed*" by test T1. Since not all the 2D lines extracted in the learning images took part in the reconstruction of the 3D scene (only those 2D lines that had a good match across all learning images), there could also be 2D lines in the test image that do not appear in the 3D scene but do appear in the learning images, and therefore should be marked as "*not-changed*". Since T1 is not able to check such cases, we apply T2.

Fig. 3. *Results of test T1.*

In T2 we look for the matching 2D line in learning image I_j which camera's center is closest to the camera's center of the test image. For every 2D line in image I_j image we calculate the two epipolar lines mapped in the test image by the fundamental matrix between the test image and image I_j. If a line segment in the test image with endpoints A and B is found, such that endpoint A's Euclidean distance to one of the epipolar line is smaller than threshold t_{2a} and endpoint B's Euclidean distance to the other epipolar line is smaller than threshold t_{2a}, and the offset between the 2D location of the two lines is smaller than threshold t_{2b}, the line is marked as "*not-changed*".

After applying tests T1 and T2 each 2D line in the test image was classified as "*not-changed*" and "*changed*", where "*changed*" in this case means that the line belongs to a new object that entered the scene.

C. *Change Detection Test for 3D Lines*

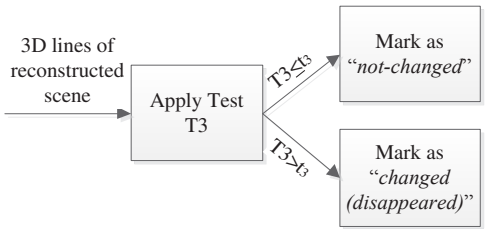

Fig. 4. *Test T3 applied to determine the "state" of each 3D line*

Test T3, as shown in Figure 4 and also used in [1], is applied on the 3D lines comprising the reconstructed scene from section IV in order to discover if an object has left the scene in the new test image. For each 3D line in the reconstructed scene we check if there exists a close enough 2D line in the test image. If such 2D line exists it means that the object did not leave the scene and that 3D line is marked as "*not-changed*". If such 2D line does not exist it means that the object has left the scene and this line is marked as "*changed*".

In order to find if a close enough 2D line exists in the test image, we use the projective camera matrix of the test image and project the 3D line to the test image. Then we use Eq. (3) to calculate the distance of the projected 3D line to all 2D lines in the test image. If the closest distance is lower than a threshold t_3 the 3D line is marked as "*not-changed*". If the minimal distance is greater than threshold t_3 then the state of this 3D line is marked as "*changed*" (disappeared). This procedure is performed for all 3D lines in the reconstructed scene.

D. *Improving Results with KNN*

As described earlier, each of the tests T1, T2 and T3 works with a user defined threshold (t_1, t_{2a}, t_{2b} and t_3 accordingly). It is obvious that in the vast majority of the cases one cannot find thresholds t_1, t_{2a}, t_{2b} and t_3 such that the change detection procedure will have no clutter (lines that were marked as "*changed*" when in fact they are not and vice versa).

Therefore, after applying the tests T1, T2 and T3 we also use the KNN algorithm to reduce this clutter. Here we use the assumption that after test T1, T2 and T3 most of the lines received the correct state and therefore if we will apply the KNN algorithm line with wrong states will receive the correct one.

In majority of the cases the lines that get the wrong state label are the ones that are marked "*changed*" when in fact they are not changed. This is due to the fact that there are more lines extracted in the test image than lines that appear in the 3D scene. Therefore, we only apply the KNN algorithm to lines that were marked "*changed*" by the previous tests. For each 2D line in the test image and 3D line in the reconstructed scene, that is marked as "*changed*", we find the closest N lines to that line (N is odd) using 2D and 3D distance metrics respectively. We give the line the state of the majority of closest lines ($> \left\lfloor \frac{N}{2} \right\rfloor$). We find that in our test images the range of N between 9-15 yields the best results.

VI. Experimental Results

We will now present the results of our method on a learning sequence of three images ($N = 3$), two different test images, and using lines matched in three images ($M = 3$). The new test images were taken from an arbitrary viewpoint (different from all the viewpoints of the learning images) and not in the same time of day, meaning different lighting conditions. We experimented with two test images – one with an object that left the scene and one with a new object that entered the scene.

Figure 5 shows the results for the test image in which an object has left the scene. Tests T1 and T2 correctly identified the new car that entered the scene and all the cars that existed in the learning images as well. In fact, the new car only entered the scene because of the rotation in viewpoint between the learning set and the test image and not because it actually moved into the scene. However, our method is not meant to distinguish between such cases and the identification is correct. Moreover, we can see that after applying tests T1 and T2 there is still some clutter on the left side of the image, where some of the lines were wrongly marked as "*changed (new)*". After applying the KNN algorithm most of this clutter was fixed (except for one line).

Figure 6 shows the results for the test image in which a new object has entered the scene. Tests T1 and T2 correctly identified both new cars that entered the scene and all the cars that existed in the learning images as well. The lower new car is in fact a new car that moved into the scene. As shown in Figure 6, our algorithm does not

distinguish between the two cars and marks them both as "*changed (new)*" correctly.

Moreover, after applying tests T1 and T2 there is still some clutter on the left side of the image, where some of the lines were wrongly marked as "*changed (new)*". After we applied the KNN algorithm most of this clutter was fixed (except for one line).

In Figure 7 it can be seen that test T3 correctly identified the newly entered car as "*not-changed*", where clutter (marked by arrows) is being dealt with using the KNN phase.

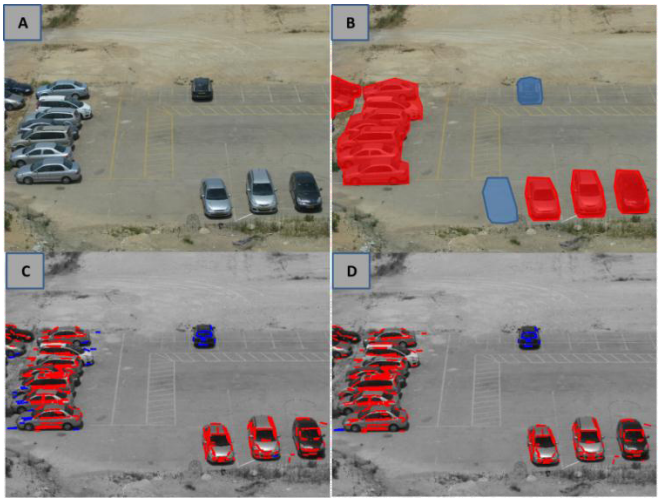

Fig. 5. *Experimental results for a test image with a car leaving the scene. (A) Test Image (B) Ground truth for test image (C) Results after T1 and T2 tests (D) Results after KNN improvement.*

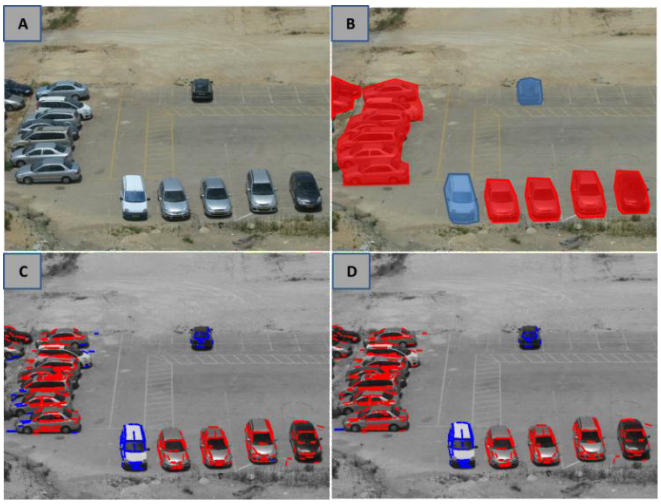

Fig. 6. *Experimental results for a test image with a car entering the scene. (A) Test Image (B) Ground truth for test image (C) Results after T1 and T2 tests (D) Results after KNN improvement.*

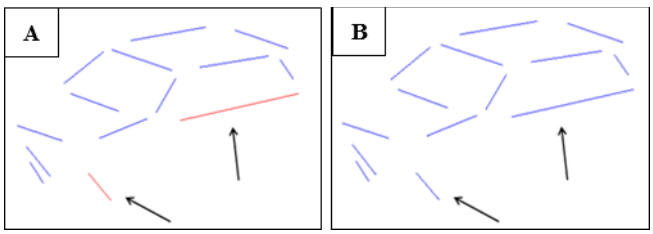

Fig. 7. *Results of T3 and KNN for the car that entered the scene, Blue lines – "changed (new)",Red lines – "not changed".(A) After T3 test (B) KNN improved results, corrected lines marked by arrows.*

VII. Conclusion

In this paper we presented a method for change detection based on 3D line segments. This method is robust and can be applied to images taken from arbitrary viewing directions, at different times and under varying illumination conditions, and to a variety of scenes. Moreover, this method does not require any prior knowledge on cameras' positions or relative distances.

Our method not only detects changes that occurred in a given test image, it also distinguishes between changes that are caused by new objects that entered the scene and changes that are caused by objects that left the scene. Our method has been shown to detect changes with high accuracy, on two change detection experiments. Those experiments indicate that our algorithm is capable of efficiently matching and accurately reconstructing small and large line segments, and detecting changes (and their types) by interpreting 2D and 3D line segments. Our 3D line segment reconstruction algorithm, which uses the geometric constraints imposed by the scene in both 2D and 3D improve the accuracy of existing individual 3D line segment reconstruction techniques.

References

[1] I. Eden, D. B. Cooper, "Using 3D line segments for robust and efficient change detection from multiple noisy images", ECCV part IV, 2008.

[2] N. C. Rowe, L. L. Grewe, "Change detection for linear features in aerial photographs using edge-finding", IEEE Transactions on Geoscience and Remote Sensing 39(7), pp. 1608–1612, 2001.

[3] W. Li, X. Li, Y. Wu, Z. Hu, "A novel framework for urban change detection using VHR satellite images", ICPR, pp. 312–315, 2006.

[4] R. Hartly, A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd Edition, Cambridge University Press, 2003.

[5] C. Baillard, C. Schmid, A. Zisserman, A. Fitzgibbon, "Automatic line matching and 3D Reconstruction of Buildings from Multiple Views", ISPRS pp. 69-80, 1999.

[6] C. Schmid, A. Zisserman, "Automatic Line Matching across Views", CVPR, 1997.

[7] D. Scharstein, "Matching Images by Comparing their Gradient Fields", *Pattern Recognition*, Vol. 1, Conference A: Computer Vision & Image Processing, 1994.

[8] L. Bruzzone, D. F. Prieto, Automatic Analysis of the Difference image for unsupervised Change Detection, IEEE Transactions on Geoscience and Remote Sensing 38(3), 1171–1182, 2000.

[9] L. Bruzzone, D. F. Prieto, An adaptive semiparametric and context-based approach to unsupervised change detection in multitemporal remote-sensing images. IEEE Transactions on Image Processing 11(4), pp. 452–466, 2002.

[10] C. Schmid, A. Zisserman, The geometry and matching of lines and curves over multiple views. International Journal of Computer Vision 40(3), pp. 199–233, 2000.

[11] C. Stauffer, W. E. L. Grimson, "Adaptive background mixture models for real-time tracking", CVPR, pp. 246–252, 1999.

[12] H. Yalcin, M. Hebert, R. T. Collins, M. J. Black, "A flow-based approach to vehicle detection and background mosaicking in airborne video", CVPR, vol. II, pp. 1202, 2005.

[13] T. Pollard, J. L. Mundy, "Change detection in a 3-d world" CVPR, pp. 1–6, 2007.

[14] A. Broadhurst, T. Drummond, R. Cipolla, "A probabilistic framework for space carving", ICCV, pp. 388–393, 2001.