Change Detection Using 3D Line Segments

By: Ido Ariav and Tom Zohar

Supervisor: Dr. Meir Bar Zohar

Spring semester, 22/05/2012

Agenda

- The Purpose of the Project
- The Problem and the Solution
- Part A - review
- Building a wire-frame model
- Change detection
- Results and Conclusions
- Future Work

Project Purpose

- Given a set of images of a certain scene (in our case – cars in a parking lot), build a 3D model of that scene, based on line segment matching of the scene in all of the images.
- The images can be taken several hours apart, from arbitrary points of view and may have different lighting conditions (linear change).
- Reconstruct wire-frame models of cars in the 3D model .
- Detect changes in a given image based on the 3D model built using the set of learning images (unsupervised).

The Problem and the Solution

- Problem – The change detection problem can become unreliable and not robust when dealing with images from multiple views and different lighting conditions.

- Solution – Compose the change detection algorithm based on a 3D model of the scene using only 3D line segment (geometric solution)
- Advantages:
  - Efficient - Detecting straight lines is computationally much less complicated then calculating correlation of all points in the images.
  - Robust - The 3D properties are independent of point of view and linear lighting conditions.
  - Reliable – a geometric solution for change detection is less sensitive to noise than one that is based on gray level comparison
  - Versatile - The use of line segments is suitable for a variety of scenarios (cars, structures, roads, etc.).

Part A - review

Algorithm for Solution

A set of training images

Find all line segments in each image using the Hough transform

Calculate Fundamental and calibrated camera matrices for each image

Line segments extracted in each image

Fundamental and camera matrices

Using epipolar constraints and cross-correlation find all corresponding lines between images

Set of lines with matches in all images

Reconstruct 3D scene using only line segments with correspondences in all 3 images

Reconstructed 3D line segments

Improve camera matrices and 3D reconstruction using a non-linear algorithm

Improved 3D line segments

Build wire-frame model and change detection – Part B

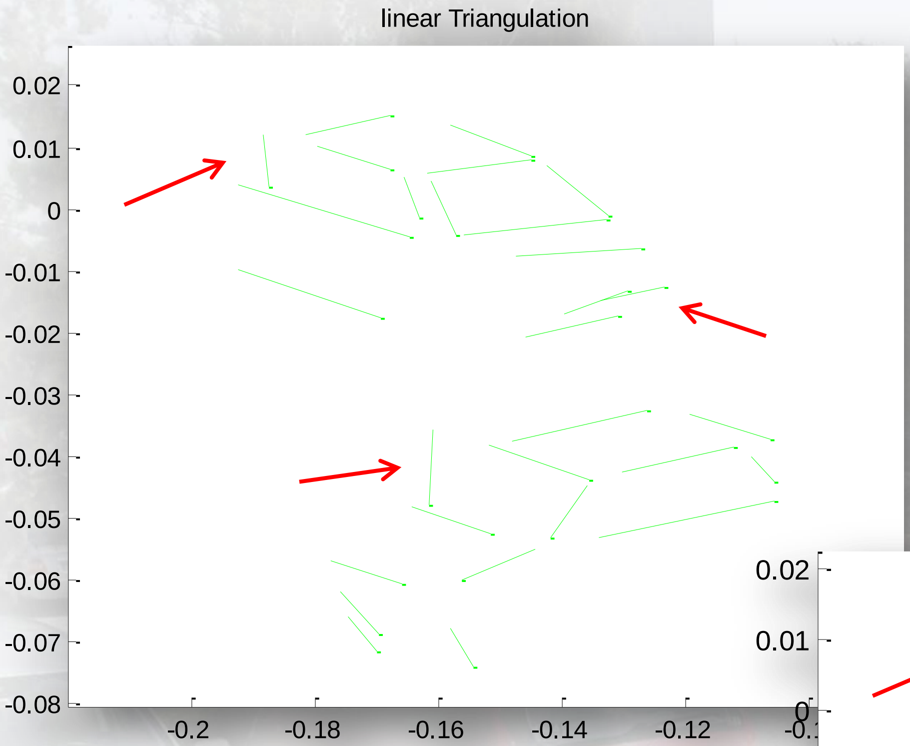
Non-Linear Optimization

- To improve our 3D reconstruction we use Nelder-Mead method to minimize the following cost function

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

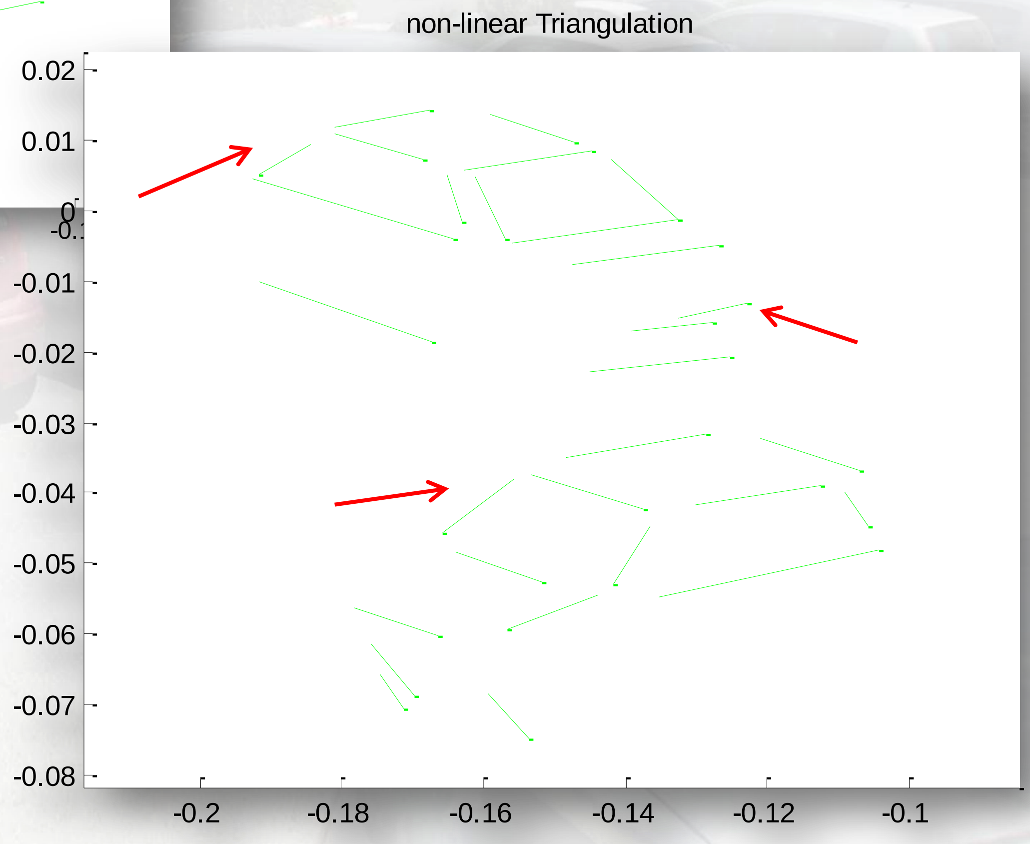
- d_l is the distance between the line and the 3D line reprojected as an infinite line.

- d_s is the distance between the line and the 3D line reprojected as a finite line.



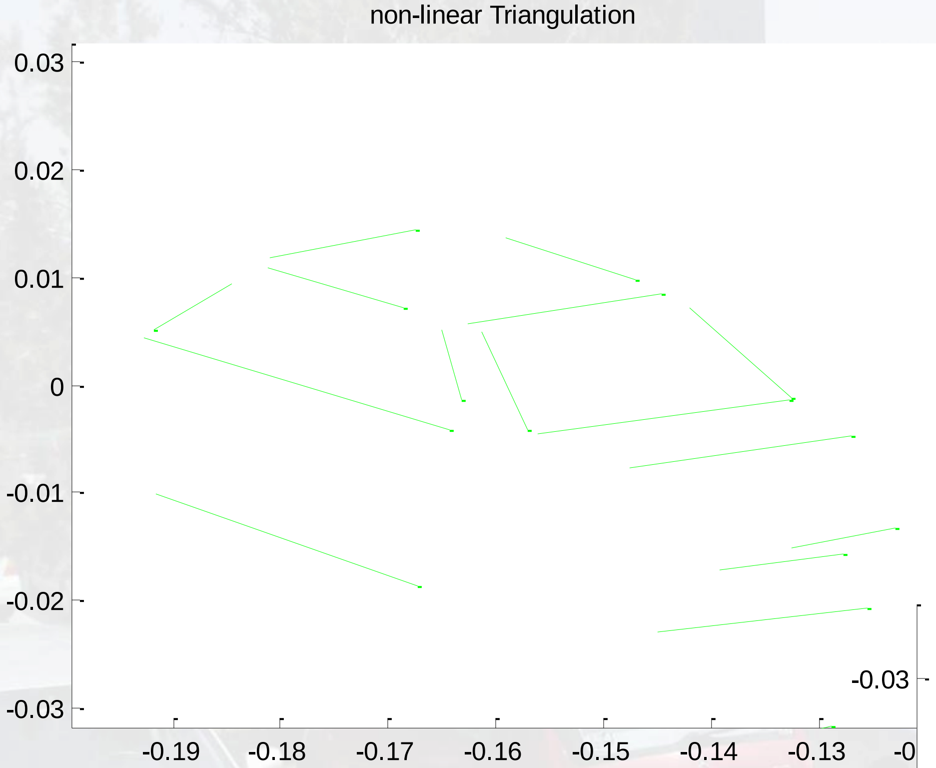
After linear reconstruction

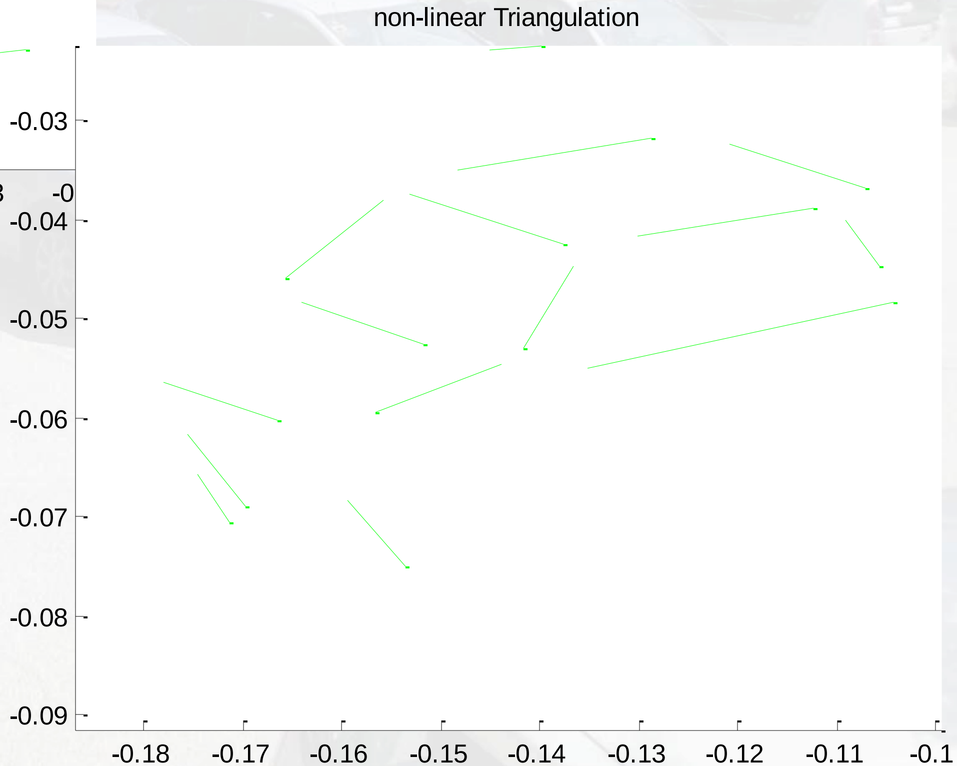
After non-linear algorithm



3 images training set

3D reconstruction after non-linear algorithm





Part B

Wire-frame models

- Use the scene's geometrical properties to link together close lines an form objects – to overcome degeneracies.
- 2 types of thresholds – in 2D and in 3D.
- Total reprojection error for all 3 views decreases.

Wire-frame models - Algorithm

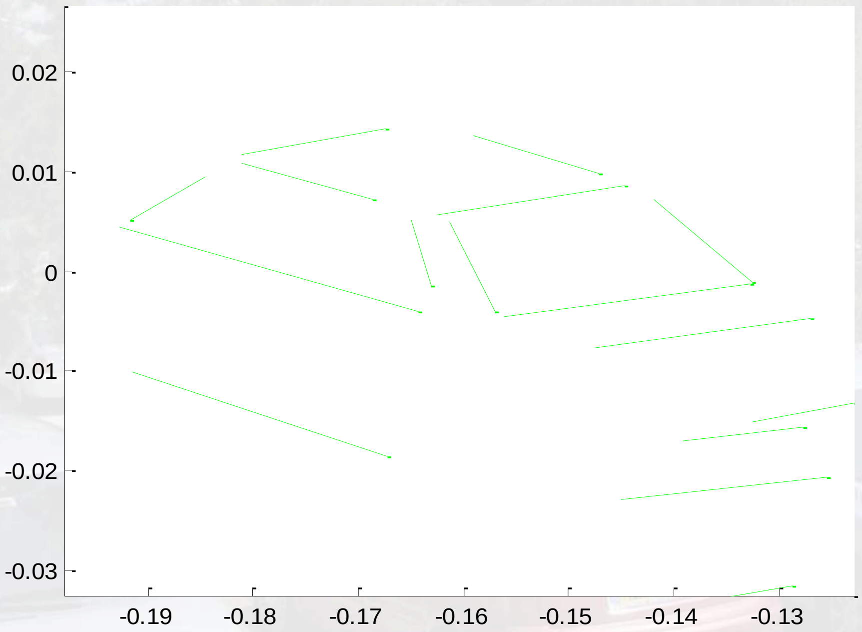
- Pick a line - l, and place it in a new object - o.
- Find a line segment l' that doesn't belong to o, which endpoint hold the closeness criterion in 3D and the closeness criterion in 2D to one of the endpoints of l.
  - For each l' found, join the 2 close endpoints by moving the endpoint of l' to the close endpoint of l. Add l' to object o.

- Solve the optimization problem for each object:

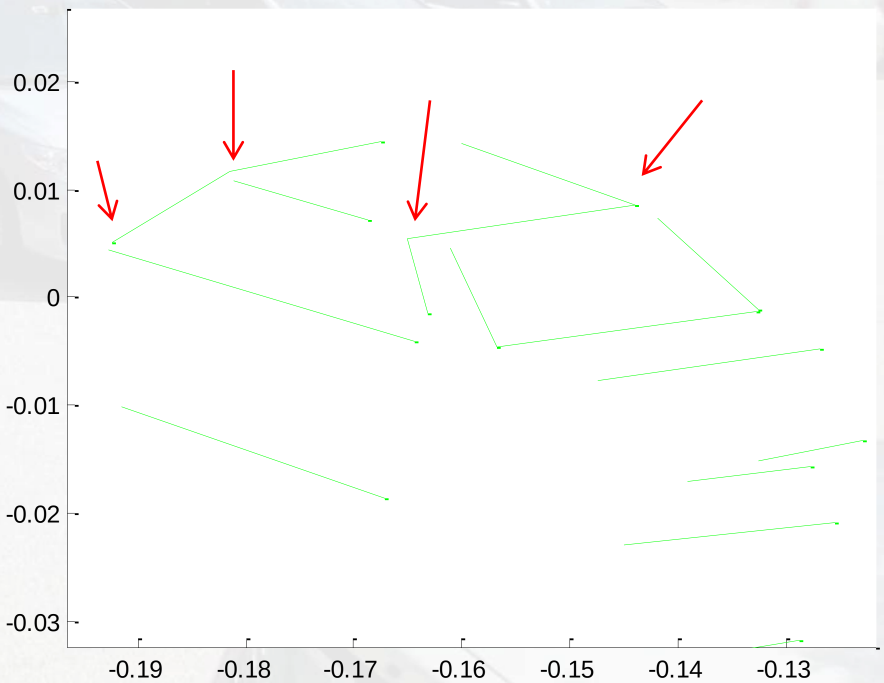
$$G^* = argmin_{V \in R^{3N_V}} \sum_{e \in E} \left(\sum_{i=1}^{n} d_l\left(l_{i_e}, l'_{i_e}\right) + \beta \sum_{i=1}^{n} d_s\left(l_{i_e}, l''_{i_e}\right) \right)$$

Where N_V is the number of vertices in a wireframe model and l_{i_e} is the line in the i^{th} image associated with edge $e \in E$ in graph $G = (V, E)$.

non-linear Triangulation



Wire Frame Model - 1 Car



Change detection

- Our goal is to correctly identify 3 types of changes –
- “Not-Changed” an object that exists both in the 3D scene and in the test image.
- “Changed (new)” an object that exists in the test image but not in the 3D scene.
- “Changed (removed)” an object that exists in the 3D scene but does not exist in the test image.

Change Detection - Algorithm

2D lines from test image

Apply Test T1

If distance to closest projection (d_s) > t1

Apply Test T2

If distance to closest projection (d_s) < t1

Mark as not-changed

If distance to closest 2D line in learning image (d_s) > t2

Mark as changed (new)

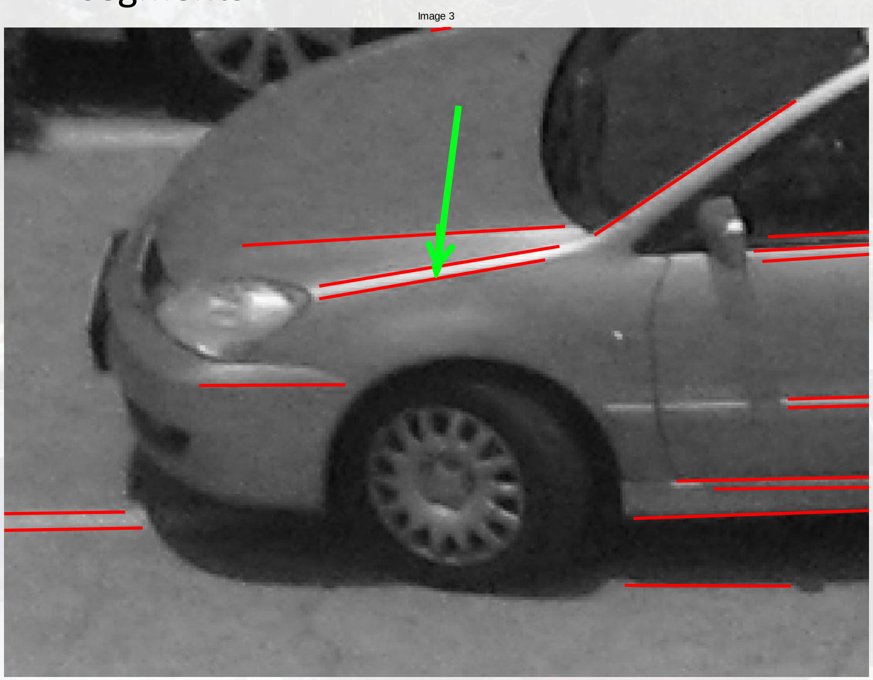
If distance to closest 2D line in learning image (d_s) < t2

Mark as not-changed

T2 - Example

Image 3 – With its estimated line segments

Test image – After T1:
Red – “not changed”
Blue – “changed”



Line segment that appears in image 3 (marked by arrow) but was not used for 3D reconstruction, exists in test in image as well (right picture).

2 epipolar beam (orange lines) mapped to the test image from the line segment in Image 3, and a 4 pixel radius threshold, t_2 (green circles) being kept by the line marked with a green arrow.

Change Detection – Algorithm cont.

3D lines from reconstructed scene

Apply Test T3

If distance of projection to closest 2D line (d_s) > t3

Mark as changed (removed)

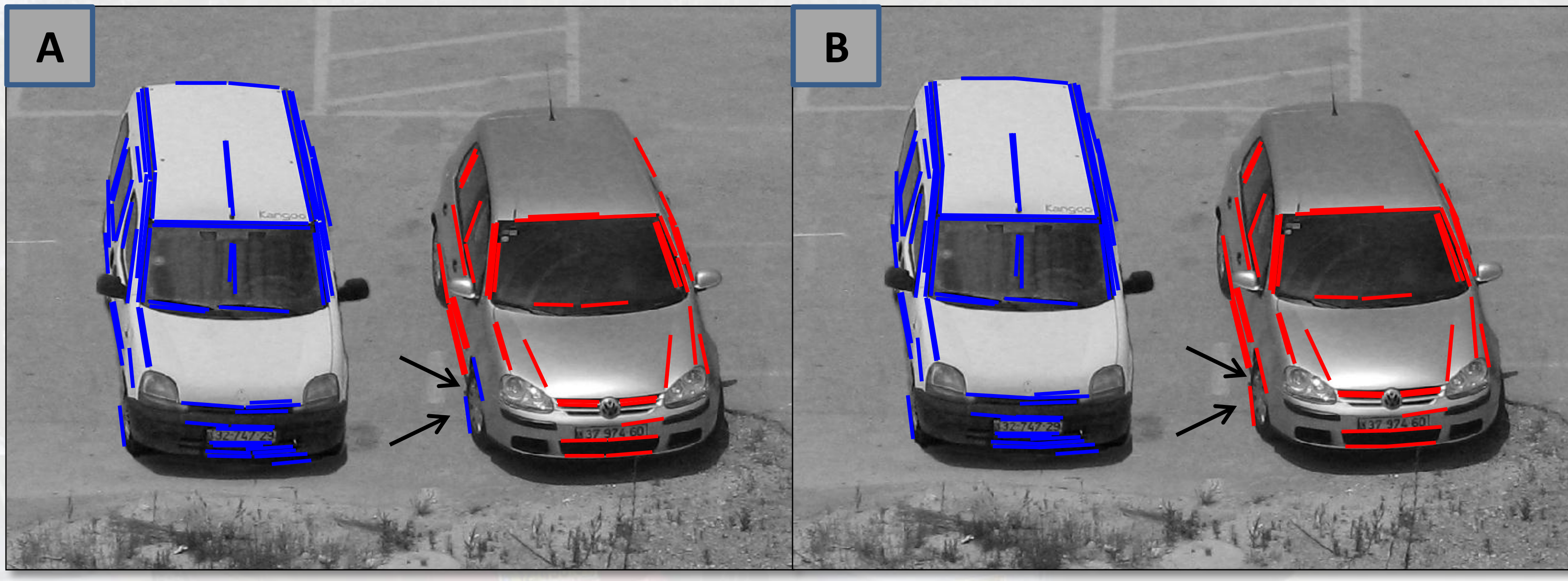
If distance of projection to closest 2D line (d_s) < t3

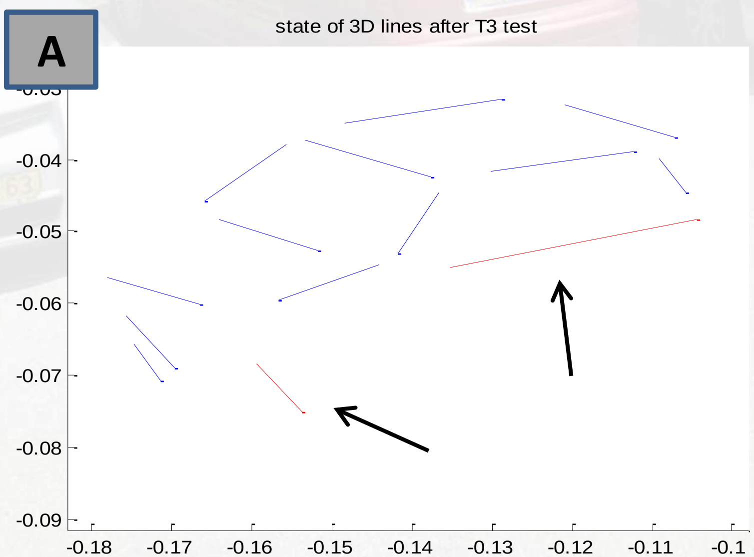
Mark as not-changed

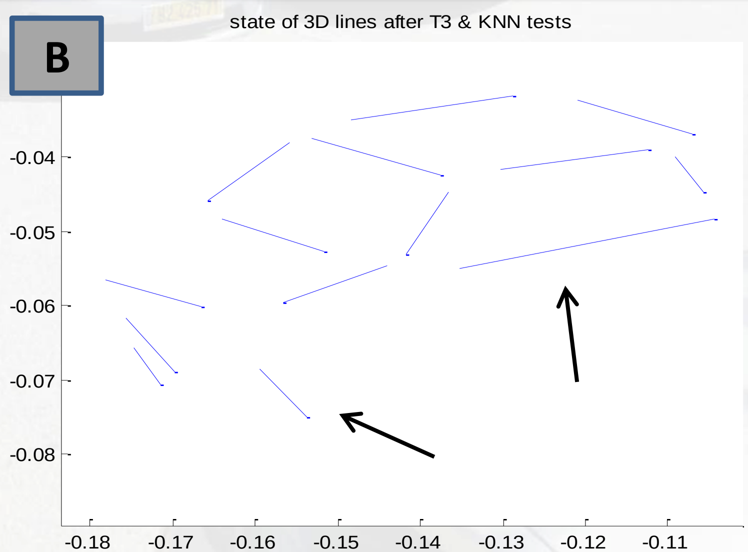
KNN Algorithm

- Improve results of lines' state (after test T1,T2 and T3) with a 2D & 3D KNN algorithm:
  1. For every 2D line in test image change state according to majority of N closest 2D lines.
  2. For every 3D line in 3D scene, change state according to majority of N closest 3D lines.
  3. Eventually we chose to work with N=15

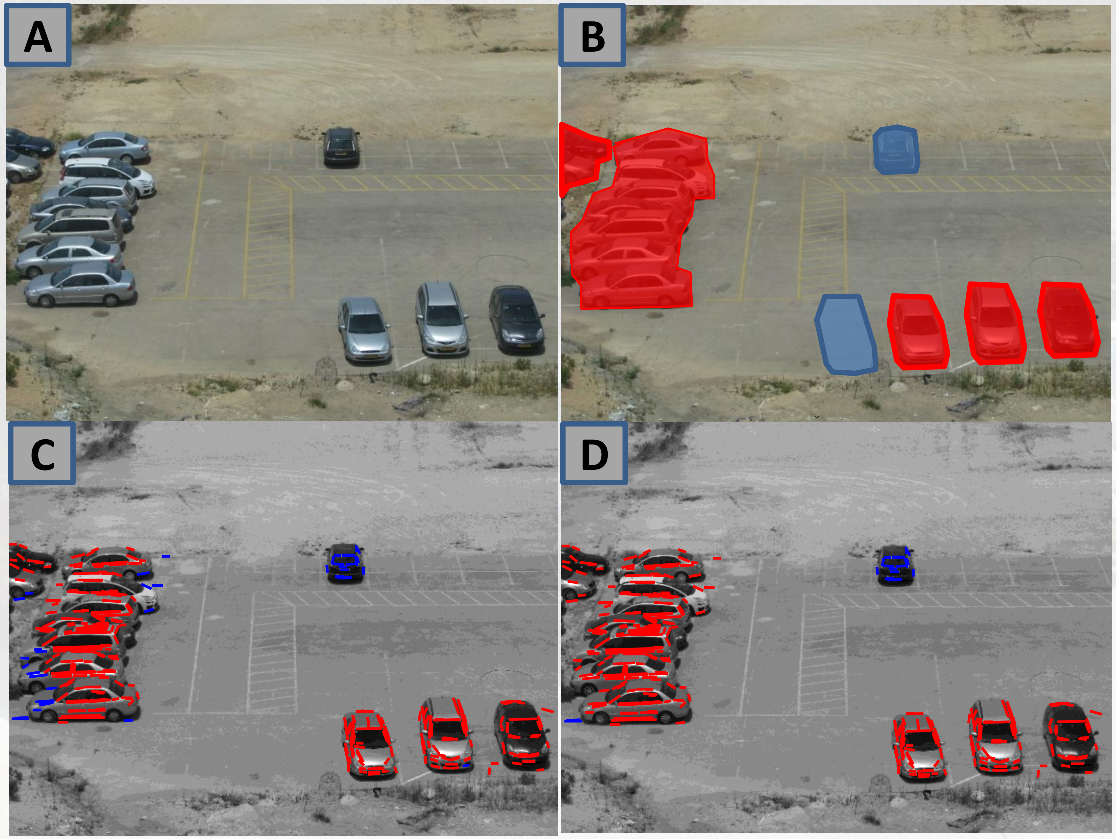
KNN Algorithm - results





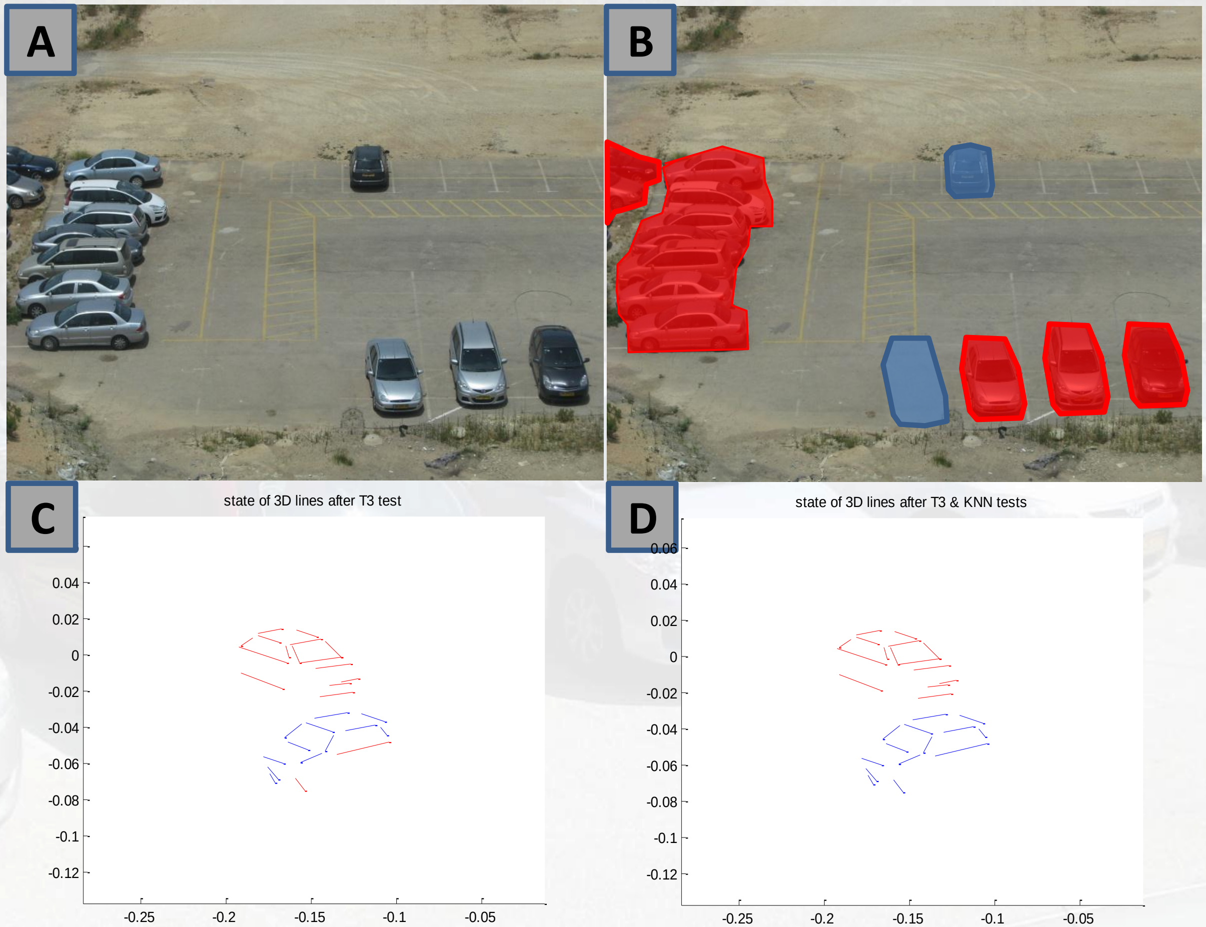


Results – Disappearance test image



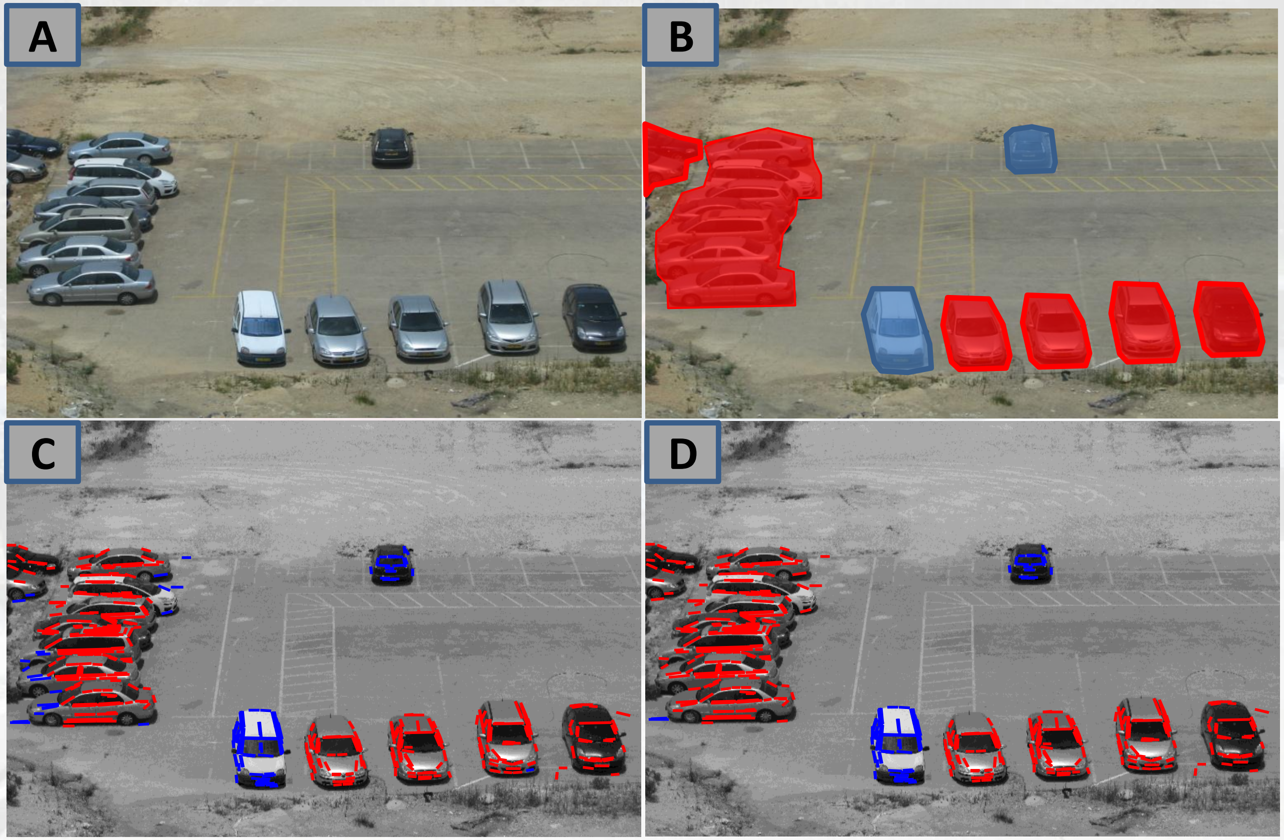
A- test image   B – “ground truth” of changes occurred in the test image
C – result after tests T1 & T2   D – results after applying the KNN algorithm

Results – Disappearance test image



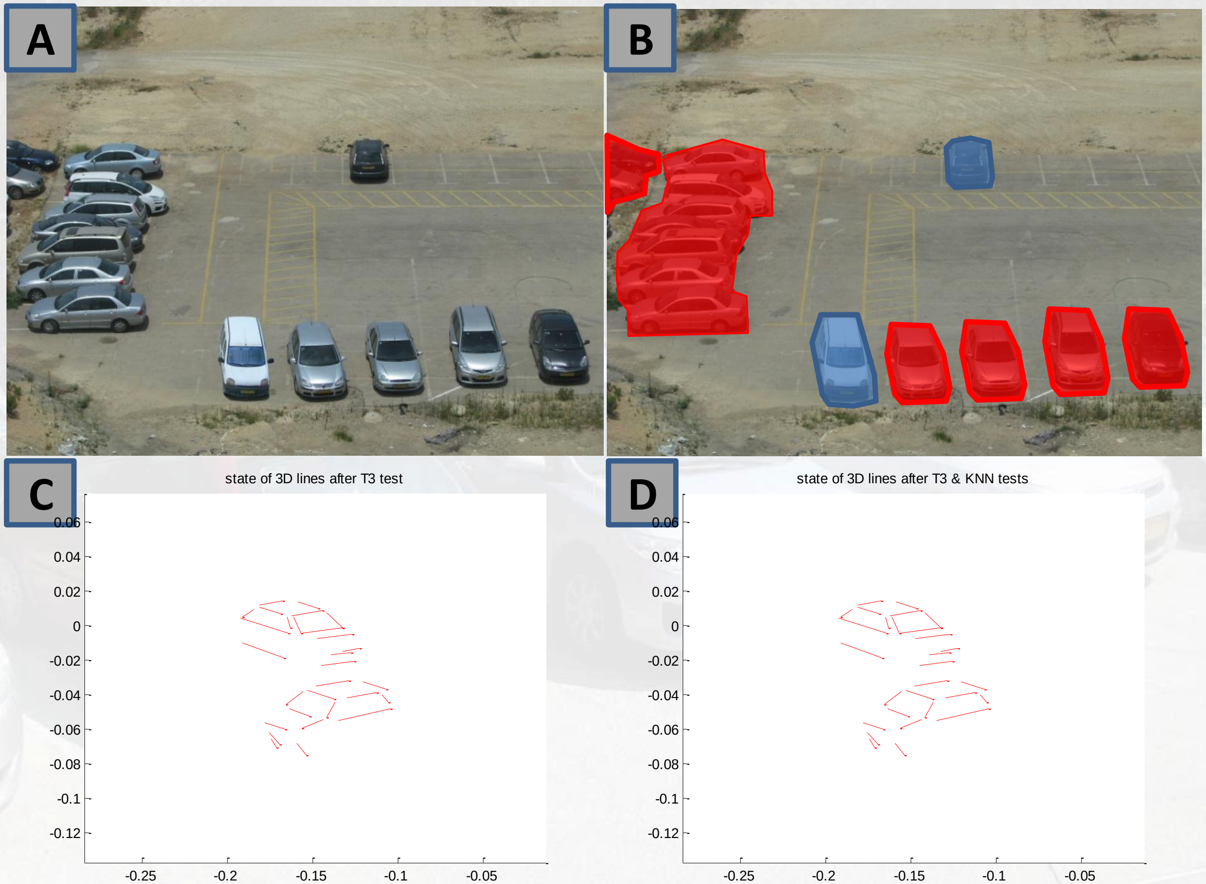
A- test image B – “ground truth” of changes occurred in the test image
C – result after test T3 D – results after applying the KNN algorithm

Results – Appearance test image



A- test image B – “ground truth” of changes occurred in the test image
C – result after tests T1 & T2 D – results after applying the KNN algorithm

Results – Appearance test image



A- test image   B – "ground truth" of changes occurred in the test image
C – result after test T3   D – results after applying the KNN algorithm

Future Work

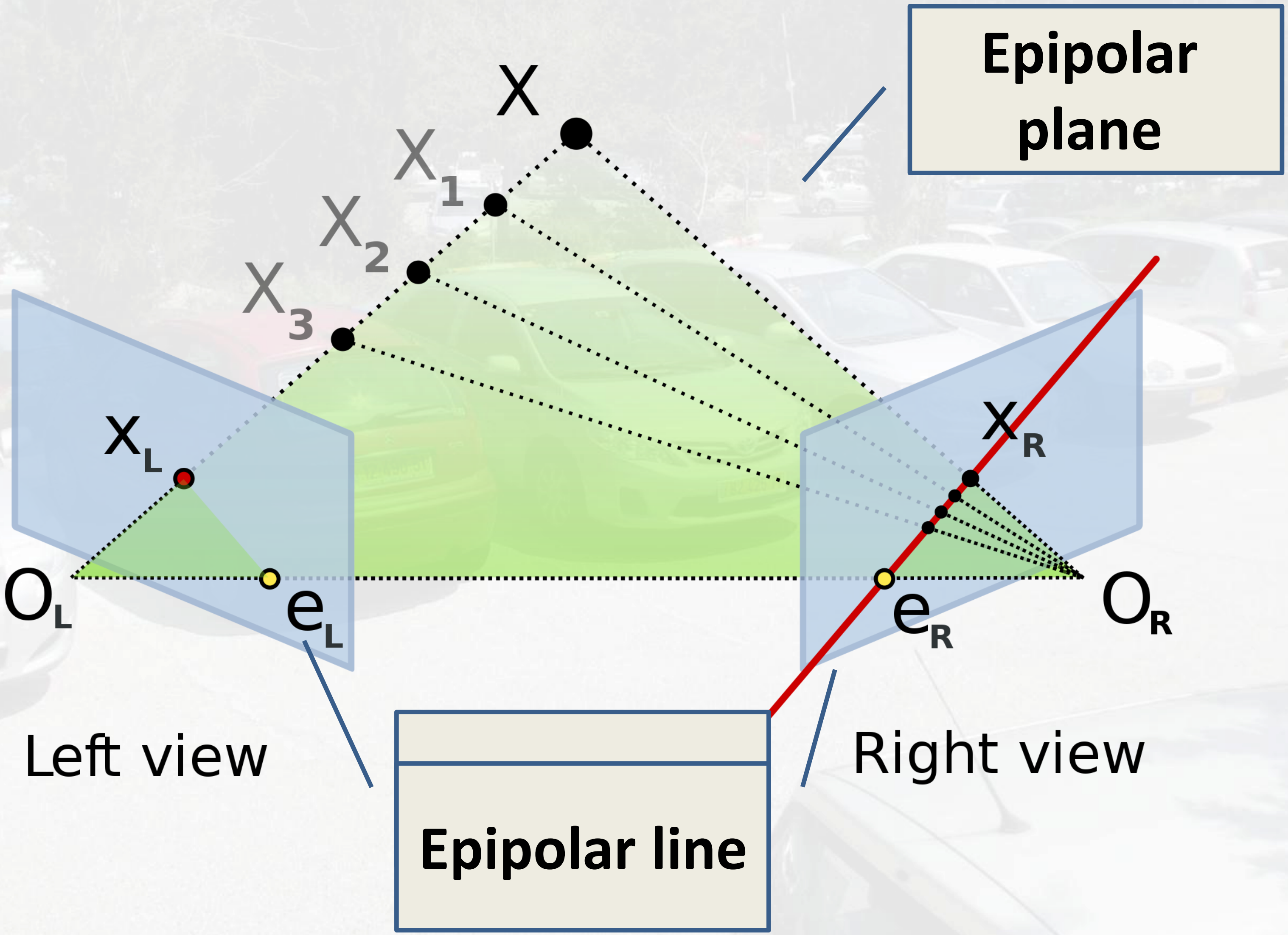
- Automation – replace all manual supervised work with unsupervised algorithms (interest point detection such as SIFT etc.)
- Object recognition – add an object recognition ability. Will help improve change detection process and overall information gain from the algorithm
- Test on other scene types – buildings, roads, aerial photos etc.

References

- R. Hartly, A. Zisserman : Multiple view geometry in computer vision, 2nd edition, Cambridge university press (2003)
- I.Eden, D.B.Cooper : using 3D line segments for robust and efficient change detection from multiple noisy images, from ECCV part IV (2008)
- C. Baillard, C. Schmid, A. Zisserman and A. Fitzgibbon : Automatic line matching and 3D reconstruction of buildings from multiple views, from ISPRS p69-p80 (1999)
- C.Schmid , A.Zisserman : Automatic Line Matching across Views, from CVPR (1997)

BACKUP

Epipolar Geometry



Line Metric

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

  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.

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

  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.