Triangulation and Segmentation-based Approach for Improving the Accuracy of Polygon Data

Alexey Noskov and Yerach Doytsher
Mapping and Geo-Information Engineering
Technion – Israel Institute of Technology
Haifa, Israel
Emails: {noskov, doytsher}@technion.ac.il

Abstract — **Often, same polygon objects are presented in Geoinformational Systems by distinct geometries with random positional discrepancies. It makes difficult to detect correspondences between data layers containing same object or parts of objects. The suggested method allows the user to improve the accuracy of one polygon layer by another more accurate polygon dataset by defining correspondences between polygons and parts of polygon boundaries. Two main techniques are applied: triangulation and segmentation. The triangulation is used to define correspondences between whole polygons by comparing triples of polygons. The segmentation approach is applied for the remaining polygons. Existing approaches do not work well in the case of partial equality of polygon boundaries. The main idea of the segmentation algorithm in this paper is based on defining correspondent segments of polygon boundaries and further replacing polygon boundary segments of the non-accurate layer with segments of the accurate data set; segments without pairs are rectified using ground control points. The resulting data contain parts of the accurate data set polygon boundaries, whereas the remaining elements are rectified according to the replaced boundary segments. From a review implemented by specialists it might be concluded that the results are satisfactory. The developed method could be applied to various types of polygonal datasets with similar scale.**

Keywords – Polyline and polygon similarity; geometry matching; shape descriptor; triangulation; topology.

I. INTRODUCTION

The same objects on different maps, which are on an equal scale, might be shown with small differences. In an ideal situation, accurate geometries of exiting maps should be used for preparing new data sets or for updating. Usually, in the real world, new maps are digitized without respect to existing data sets. Using geometries (e.g., river line) from an accurate topographic map for creating a thematic map (e.g., soil map), in many cases, is better than digitizing a new element. Often, data are unavailable, or available with significant restrictions, because of legal, technical, or other reasons. Additionally, even if an accurate data set is freely available, people usually do not want to spend time using an existing data set; in most cases they prefer to digitize new geometries on a satellite image or scanned map. These data should be aligned using accurate data sets [1]. This problem is especially sensitive for large-scale maps and plans [2].

The problem which is described in the paper refers to cadastral and city planning maps. A cadastral map is a comprehensive register of the real estate boundaries of a country. Cadastral data are produced using quality large-scale surveying with Total Stations, Differential Global Positioning System devices or other surveying systems with centimeter precision. Normally, the precision of maps based on non-survey large-scale data (e.g., satellite images) is lower. City planning data contain proposals for developing urban areas. Most city planning maps are developed by digitizing handmade maps, using space images. Almost all boundaries have small discrepancies in comparison to cadastral maps. It is very important to use exact boundaries, or their segments, on city planning data from a cadastral map, especially in central parts of cities. The approach described in the paper enables us to resolve this problem of matching two data types. Rectifying data using a set of ground control points is a popular way of improving the accuracy of a map [3]. The results of this approach are not satisfactory in many cases, because rectified objects cannot be identical to directly measured accurate objects. Another possibility is based on defining correspondent objects on an accurate data set by geometry or attributes and replacing objects from the non-accurate set with the accurate correspondent objects [4].

We present a triangulation approach. It enables us to define correspondent polygons of two datasets. It is achieved by dividing polygons into triples and comparing the triples of two datasets. A serious problem with this approach follows from the fact that objects could be partially similar (e.g., some segments of a polygon boundary are same, other parts are different). In contrast to existing approaches, the main idea of a segmentation approach is based on defining correspondent segments of polygon boundaries and further replacing polygon boundary segments of the non-accurate layer with segments of an accurate data set; segments without pairs are rectified by ground control points. The segmentation complements the triangulation algorithm. Triangulation is a fast process for defining correspondences between whole polygons. Segmentation is much slower. It enables us to define correspondences between boundary segments of polygons (excluding polygon pairs defined by triangulation). The triangulation is also used for evaluating results. The proposed algorithm could be applied to different sorts of polygon datasets with small boundary differences. The approach has not only been designed for city planning and cadastre datasets.

This paper is structured as follows: the related work is considered in Section II. Source datasets and the process of

defining initial variables are described in Section III. The triangulation approach is proposed in Section IV. The algorithm of defining correspondent polylines is presented in Section V. The process of compiling the final map is described in Section VI. The results are discussed in Section VII. The conclusion is presented in Section VIII.

II. Related Work

In order to develop the proposed algorithm, various approaches were considered. The review of these approaches is presented in this section. Many of them were evaluated. We found several useful concepts for our task described in the considered papers. The papers are grouped. The groups appear in the order in which they influenced our research. Most of our ideas were taken from the feature-based matching group of approaches. The relational matching ideas also affected our approach, mainly in the sense of topological orientation of the developed approach. We have not found useful concepts for the context of the discovered datasets in the last category (attributes-based matching), but it discloses and raises many useful problems of attribute processing for data matching researchers. Additionally, several programming techniques are described at the end of the section, in order to improve the quality of the developed approach. Most of the techniques are applied.

Discrepancy problems on digital maps can be resolved in different ways. Common shape matching techniques are currently used in the raster and vector fields, and sometimes in combination with each other. Several common techniques in the field of Shape Similarity or Pattern Recognition could be applied to the various needs of the matched objects and relevant research questions.

Vector matching techniques can be divided into three main categories.

A. Feature-based matching

This group of methods is based on an object's geometry and shape. The degree of compatibility of objects is determined by their geometry, size, or area. The process is carried out by structural analysis of a set of objects and comparing whether similar structural analysis of the candidates fits the objects of the other data set [5][6]. In [4], comparison of objects is based on analysis of a contour distribution histogram. A polar coordinates approach for calculating the histogram is used. A method based on the Wasserstein distance was published by Schmitzer et al. [7]. A special shape descriptor for defined correspondent objects on raster images was developed by Ma and Longin [8]. Feature-based matching approaches do not allow for resolving our problem, because they have been developed mainly for single shapes; however, we can use them as part of our approach.

B. Relational matching

This group of methods takes objects' relationships into account. In [9], topological and spatial neighborly relations between two data sets, preserved even after running operations such as rotation or scale, were discovered. In relational matching, the comparison of the object is implemented with respect to a neighboring object. We can verify the similarity of two objects by considering neighboring objects. The problem of non-rigid shape recognition is studied by Bronstein et al. [10]; the applicability of diffusion distances within the Gromov-Hausdorff framework [10] and the presence of topological changes have been explored in this paper. A multiple-point geostatistical modeling based on cross-correlation functions is proposed by Tahmasebi et al. in [11].

C. Attributes-based matching

Matching two data sets' objects by attributes could be very effective if a similar data model is used. Two types of attribute matching could be mentioned: Schema-based [12] and Ontology-based. The concept of semantic proximity, which is essentially an abstraction/mapping between the domains of the two objects associated with the context of comparison, is proposed by Kashyap and Sheth in [13]. In [14], an approach based on both types is presented. An ontology-based integration of XML Web Resources focusing on the significance of offering appropriate high-level primitives and mechanisms for representing data semantics is described by Amann et al. in [15]. A technique for building approximate string join capabilities on top of commercial databases by exploiting facilities already available in them is described by Gravano in [16] and [17]. Attributes-based matching is a specific group of approaches; it can only be applied efficiently in special cases with special data. In most situations it is ineffective.

The merging and fusion of heterogeneous databases has been extensively studied, both spatially [18] and non-spatially [19]. The Map conflation method is based on data fusion algorithms; the aim of the process is to prepare a map which is a combination of two or more maps (often for updating an old map). Map conflation approaches are presented in [2] [20] [3]. In [21], three approaches for the linking of objects in different spatial data sets are described. The first defines the linking as a matching problem and aims at finding a correspondence between two data sets of similar scale. The two other approaches focus on the derivation of one representation from the other one, leading to an automatic generation of new digital data sets of lower resolution.

In order to resolve the described problem, the mentioned approaches have been considered. It has been concluded, that a new solution need to be developed.

Computer Vision algorithms are popular in the field of data matching [22]. The Open Computer Vision (OpenCV) framework [23] is widely used today; it provides a number of "out-of-the-box" functions enabling us to detect and compare objects and bindings for popular programming languages (e.g., Python [24]). This makes the OpenCV framework very useful for data-matching tasks. Delaunay Triangulation [25] and Voronoi Polygons [26] are very useful techniques for working with discrete vector data and neighbor analysis. We should also note that in practice, data is distributed in non-topological formats (e.g., Shape File format). That leads to complication of the analysis, because of a surplus number of objects and duplication of primitives,

e.g., polygon boundaries, unexpected gaps between objects etc. We need to use one of the topological data formats presented by Landa [27] to avoid these obstacles. The topology in GIS context is described in detail by Blazek et al. in [28]. The main topological data types are presented: point, line (comprising nodes, vertices and segments), and polygons (consisting of boundaries and centroid). Many useful GIS definitions and techniques, including geometry relations, topology and operations (e.g., overlay), are described by Herring in [29].

Additionally, two perspective methods could be used in GIS data matching to reduce the time and computer resources required: Genetic Algorithms [30] help to avoid Brute-force operations in some cases; OpenCL technology [31] makes it possible to split a process into a huge number of parallel threads on a video card.

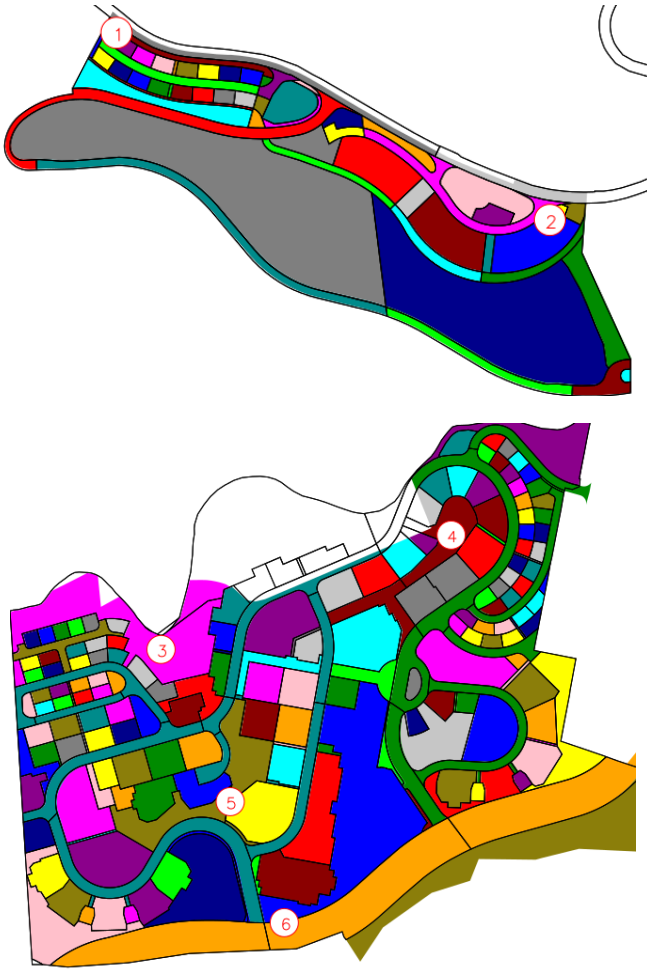

Figure 1. Source data: land-use city planning (color background) and cadastre (black polylines) of Nesher (upper) and Yokne'am (lower) datasets.

III. DEFINING INITIAL VARIABLES

For implementing and testing our approach, GIS data provided by Survey of Israel have been used. They contain cadastre and land-use city planning polygon shape files covering a part of Nesher and Yokne'am (towns in the Haifa District of Israel). Figure 1 depicts source data; red numbers in circles correspond to numbers of extents in Figure 21. Overlaid polygon boundaries of two data sets are presented in Figure 2. From the figure, one can conclude that transformation of lines would not yield positive results, because the gaps are extremely variable - the curved parts of lines consist of different numbers of vertices; thus, even with correct parameters of transformation, the result would not be satisfactory.

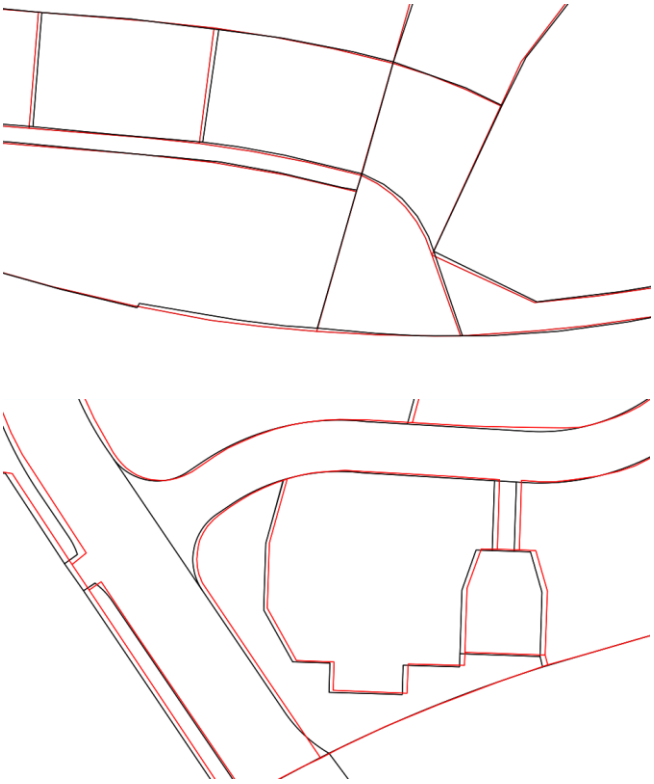

Figure 2. Positional discrepancies of city planning (red lines) and cadastre (black lines) datasets: Nesher (upper) and Yokne'am (lower).

Source shape files have been converted to GRASS GIS 7 topological data format [27]. Data preparation can be divided into 3 steps:

- Extracting polygon boundaries.
- Splitting polylines into a set of equidistant points. For depicting this parameter we will use the symbol d in the paper.
- Calculating an array of distances between the nearest points of two datasets. Setting of initial measures.

We have decided to use 2 meters between equidistant points. Using a greater distance makes impossible to detect small curves, whereas a smaller distance significantly increases the calculation time.

Several initial measures need to be calculated. Maximal distance (D_{max}) between the nearest points of two datasets

and maximal standard deviation (σ_{max}) have been calculated. To calculate these parameters we need to create a list of 100 percentiles. Then we implement a loop from the first to the last percentile on the list. D_{max} equals percentile i and σ_{max} equals the double standard deviation of distance in the interval between percentiles number i and 100 if the standard deviation of distances between percentiles i-1 and i is more than 1. We calculate tail parameter (t) as follows: $t= D_{max}/d$, minimal tail parameter equals 4. Tail defines a starting or ending segment of polyline that can be ignored.

We have developed a special shape descriptor (S), based on the descriptor presented in [8]. The descriptor measures the similarity of polylines. Polylines are more similar if S is larger.

$$d = \log_{10}(1 + distA) - \log_{10}(1 + distB)$$
$$S = \frac{\sum \exp\left(-d^2 + \exp(-(angA - angB)^2)\right)}{(k \cdot k)^2} \quad (1)$$

In the equation, 1 means matrix of ones, distA – matrix of distances between all pairs of points laid on polyline a. distB – matrix of distances between all pairs of points laid on polyline b. If the number of points of a line is k, then matrix size is k×k. angA and angB are matrices of angles in radians between all pairs of points of lines a and b, correspondingly. The shape descriptor is calculated for the segments with equal length (k).

A list containing pairs of point sets has been prepared, where all points laid on line A are closest to points laid on line B of another dataset. For each element of the list, two shape descriptors of tails with t number of points have been calculated and collected into a list of shape descriptors of tails. St_min, St_max – minimal and maximal elements of the list. Also, we use maximal tail standard deviation of point distances (σt), and its (maximal tail) maximal value – σt_max.

The list of initial variables has been calculated:

Nesher datasets: Dmax=2.1, σmax=1.0, St_min=0,St_max=0.23; Yokne'am datasets: Dmax=7.9, σmax=1.5, St_min=0,St_max=0.25.

IV. DEFINING CORRESPONDING POLYGONS OF DATASETS BY TRIANGULATION

As we can see in Figure 1, many polygons of city planning datasets have corresponding polygons in cadastre datasets. Thus, we can simply take attributes of these city planning polygons and link them to the geometry of correspondent cadastre polygons. To implement this idea we have developed a triangulation algorithm.

The triangulation consists of several stages: calculating of Delaunay triangulation based on polygon centroids; comparing all possible pairs of polygon triples and defining correspondent candidate pairs of polygon triples of two datasets by area and perimeter comparison; defining correct triple correspondence by considering distances between polygon centroids. Further in this section, the algorithm will be described in detail.

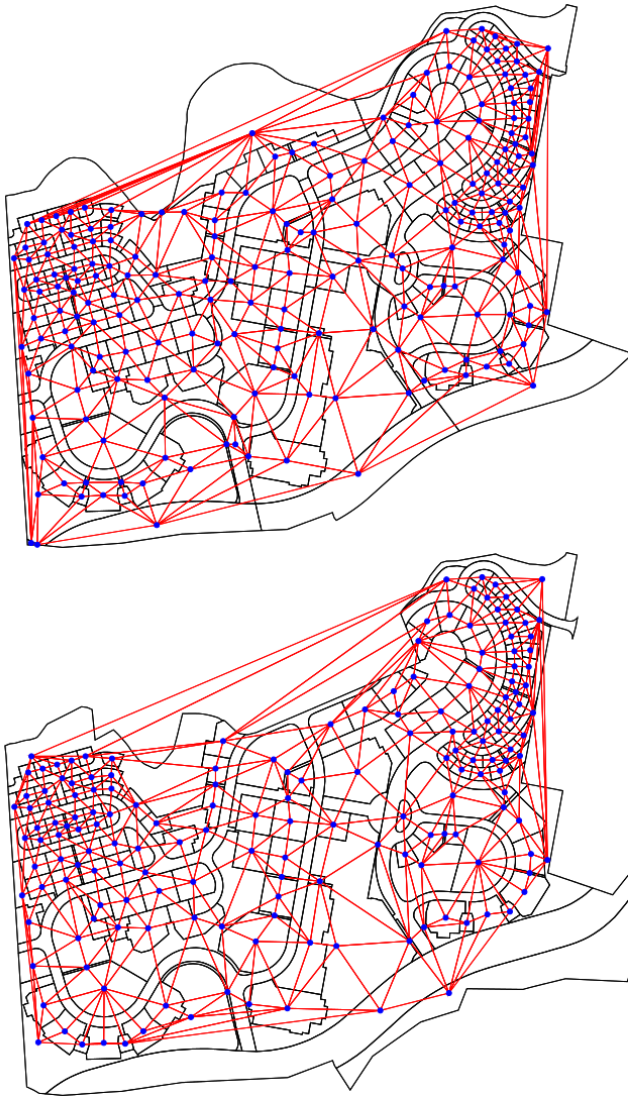

Figure 3. Delaunay triangulation (red lines) of cadastre (upper) and city planning (lower) datasets. Black lines are polygon boundaries, blue points are polygon centroids. Yokne'am.

Delaunay triangulation maps are presented in Figure 3. These maps enable us to select triples of polygons, where each triple belongs to one of the triangles (centroids of polygons are vertices of Delaunay triangles). Then, for each triple we try to find a candidate counterpart triple on a second dataset. A candidate counterpart is detected by comparing perimeters and areas of polygons as follows. A triple of first dataset polygons is the candidate counterpart of a triple of a second dataset, if for each polygon in the first triple we can find a polygon in the second triple (each polygon can only participate in one correspondence). The area and perimeter of the first polygon are more than the area and perimeter of the second polygon minus 20% and less

than the area and perimeter of the second polygon plus 20%. The value 20% is empiric. It has been defined as optimal for the considered datasets, but it may be used with other datasets. It could be feasible to let the user modify the value.

At this time a list of candidate counterpart triples is prepared. The candidates are evaluated by distances between centroids of counterpart polygons to define the most likely correspondence of polygons in the triple pair (TripleB.GetTheMostSimilar() function in Figure 4). For each candidate triple of polygons, all possible combinations of polygon correspondences implemented by permutation are considered. For each combination, a sum of distances between correspondent polygons is calculated. A combination with the lowest sum describes the most likely correspondence of polygons in the current triple pair. The correspondence of polygons in the triple pair is correct if one of the two follows conditions is true. The first condition is valid if the sum of distances between centroids of correspondent polygons is less than D_{max} defined in the previous section. This condition does not work for incompact long curve polygons (e.g., road polygons), because even small changes in polygon boundary significantly changes centroid position. That is why another condition has been developed. This condition is valid if the mean distance between boundaries of all correspondent polygons in the triple pair is less than D_{max}. To calculate the mean distance between polygon boundaries, the boundaries have been split by equidistance points with intervals equaling d (2 meters) as defined in the previous section. For each point of the first polygon boundary, a distance to the closest point of the second polygon boundary is calculated. The mean distance between the polygon boundaries equals the mean value of the calculated point distance.

The described algorithm is presented as a pseudo code listing in Figure 4 and Figure 5. Figure 4 explains the process of preparing a candidate counterpart triples list. It mainly comprises standard geoprocessing operations like buffering, triangulating and overlaying. In order to improve the performance of the algorithms, we used a number of tricks for calculating a list of candidate counterpart triples, presented in Figure 4. In Figure 5, we consider a process of evaluating the candidate list by a number of conditions and loops. Each element of a result list contains polygon pairs of two datasets. The following text contains more detailed explanation of the listings.

Figure 4 starts from the definition of source polygon maps. PolsA and PolsB compare polygon maps. For each map several preparatory procedures are applied. Area, perimeter and centroid coordinates have been added to the attribute table for each polygon. Then, a map of centroids is created. An attribute table of source polygon is inherited. GetBuffer function returns the 0.1 m buffers around centroids. In our case, 0.1 m means the small value that we can ignore, i.e., we consider it as “almost 0”. Another small value could be used; it depends on specific datasets and software. The attribute table is also inherited. GetDelaunay pseudo function generates a triangulation map based on centroids. OverayMap is the result of overlaying the buffer and triangulation map with an “and” operator. The three last described operations are illustrated in Figure 4. This approach of grouping polygons into triples works very fast. Many GIS applications have the described functions in the standard edition.

```
PolsA=first_polygon_map
PolsB=second_polygon_map
InitTriples=GetEmptyList()

Foreach map in [PolsA, PolsB] {

   CalculateAreaOfPolygons(map)
   CalculatePerimeterOfPolygons(map)
   CalculateXYOfPolygonsCentroids(map)
   CentroidMap=GetCentroidsAsPoints(map)
   BufferMap=GetBuffer(CentroidMap, buffer_size=0.1)
   TriangMap=GetDelaunay(CentroidMap)
   OverlayMap=GetOverlay(BufferMap, TriangMap)
   TempList=GetEmptyList()

   Foreach TriangleId in GetIds(TriangMap) {

     Attributes=GetAttribs(OverlayMap,get=map_perimeter,
map_area,map_centroidXY, where=TriangMap_id= TriangleId)
     TempList.append(Attributes)

  }

   InitTriples.append(TempList)

}

CandidateTriples= GetEmptyList()
Foreach TripleA in InitTriples[0] {
  Foreach TripleB in InitTriples[1] {
    Appropriate=True
    Foreach PolA in TripleA {
      PolB=TripleB.GetTheMostSimilar(PolA)
      TripleB.remove(PolB)

      If not (0.8*PolB.area < PolA.area < 1.2*PolB.area) and not
(0.8*PolB.perimeter < PolA. perimeter < 1.2*PolB. perimeter) {
          Appropriate=False
       }

    }

    If  Appropriate==True {
       CandidateTriples.append([TripleA,TripleB]) }
    }
```

Figure 4. The first part of the triangulation algorithm: preparing of a list of candidate counterpart triples.

Now we can easily get a polygon triple belonging to any triangle. The first element of InitTriple contains all triples of polygons of the first polygon map; the second element contains all triples of polygons of the second polygon map. Then the triples of the two polygon maps are compared by area and perimeter and appropriate triples are added to the CandidateTriples list.

In Figure 5, an algorithm for evaluating candidate counterpart triples and defining correspondences between polygons is described. The combinations list comprises all possible correspondences of polygons in a triple pair. For each combination, a sum of distances between the centroids of correspondent polygons is calculated. A combination with a minimal sum of distances is kept in optimal variable for further processing.

```
Result= GetEmptyList()

 Foreach TripleA, TripleB in CandidateTriples {

  Combinations=getAllPossibleCombinations(TripleA, TripleB)
   DistList= GetEmptyList()

   For {i=0; i<length(Combinations);i++} {

        PolA1,PolB1,PolA2,PolB2,PolA3,PolB3=
Combinations.GetPolygonsAsList()

        DistList.append(  CentoidDistance(PolA1,PolB1)+
CentoidDistance(PolA2,PolB2)+ CentoidDistance(PolA3,PolB3)+)

   }

   MinDist= Minimal(DistList)
   Index=DistList.GetIndex(MinDist)

   Optimal= Combinations[Index]
   Foreach PolA, PolB in Optimal {

     If not (0.8*PolB.area < PolA.area < 1.2*PolB.area) and not
(0.8*PolB.perimeter < PolA. perimeter < 1.2*PolB. perimeter) {
        Continue
}

  Appropriate=True

   If MinDist < max_dist {
     Appropriate=True
   }

   Else {
    Foreach PolA, PolB in Optimal {
      Xa,Ya=PolA.CentroidCoords
      Xb,Yb=PolB.CentroidCoords
      AreaA=PolA.area
      Delta=Sqrt(AreaA)+max_dist

      If (Xa-Delta < Xb <Xa+Delta) and (Ya-Delta < Yb
<Ya+Delta) {

        EqdBoundsMapA= GetEquidistancePoints(PolA)
        EqdBoundsMapB= GetEquidistancePoints(PolB)
        DistArray=PointDistance(EqdBoundsMapA,
EqdBoundsMapB)

        If Mean(DistArray)>max_dist {
           Appropriate=False
        }

       }
       Else {
           Appropriate=False
       }
    } #end of Foreach PolA, PolB in Optimal

    If Appropriate == True {

      PolA1,PolB1,PolA2,PolB2,PolA3,PolB3=
Combinations.GetPolygonsAsList()

       Result.append([PolA1,PolB1])
       Result.append([PolA2,PolB2])
       Result.append([PolA3,PolB3])

   }
```

Figure 5. The second part of the triangulation algorithm: preparing a list of polygon correspondences.

All polygon correspondences are evaluated using areas and perimeters; if the condition is false, the triple pair is not considered and the next candidate is processed. If the sum of distances is less than D_{max} (max_dist in the listing; the parameter has been defined in the previous section), the polygon correspondences are correct and are added to the Result list. If the previous condition is false, then we calculate mean distance between polygon boundaries. It is implemented by calculating distances between equidistant points splitting boundaries. If the mean distance is less than D_{max}, a polygon pair passes the check. This condition has to be true for all polygon pairs in the combination. The performance of the described condition is quite low, which is why we use the Delta variable for filtering distant polygons. Delta equals to the square root of PolA's area plus D_{max}. Only if centroid PolB is placed inside a rectangle Xa-Delta, Ya-Delta, Xa+Delta and Ya+Delta (where Xa and Ya are PolA's centroid coordinates), we can calculate mean distance between polygon boundaries.

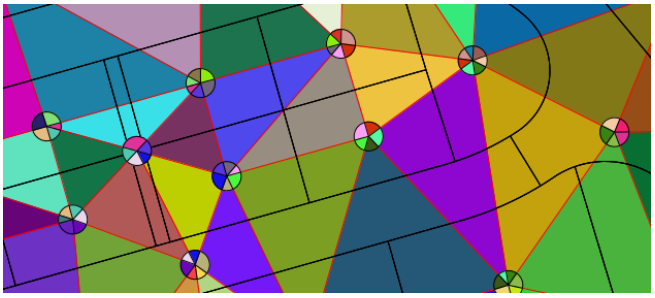

Figure 6. Grouping polygons into triples by triangulation (color background and red boundaries), buffering (circles) and overlaying (color sectors of the circles).Yokne'am.

The cadastre and city planning polygon datasets have been compared by the triangulation algorithm, and counterpart polygons have been detected. The result is presented in Figure 7 and in Figure 8: counterpart polygons are depicted by gray background and black boundaries, polygons without pairs are light gray areas with gray boundaries.

Almost all polygons of the Nesher datasets (see Figure 8) have correspondences; only several northern polygons do not have pairs. Many polygon correspondences have been defined on the Yokne'am datasets (see Figure 7). Several polygons look similar in the figure and have no defined correspondences. That means that either we do not see the differences because of the scale, or that polygons participate only in incorrect triples (triples with at least one polygon without correspondences).

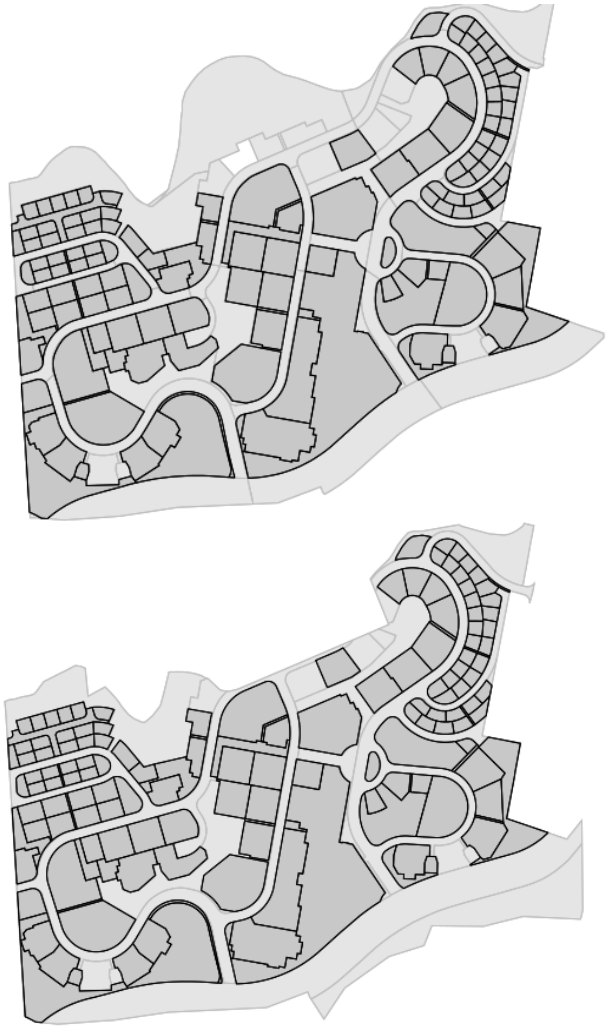

Figure 7. The result of triangulation. Upper – cadaster, lower – city planning. Yokne'am.

The counterpart polygons are excluded from further processing and will be involved in the processing only at the last stage of the approach. The polygons without pairs are extracted from the datasets for defining correspondences between boundaries and boundaries' segments.

V. DEFINING CORRESPONDING LINES OF DATASETS

To define corresponding lines, we have developed a special descriptor based on several measures: distances between points, standard deviation of distances, shape descriptor. Figure 9 depicts the main idea – using equidistant points on a polyline to detect corresponding polylines, or segments of polylines. In the figure, a polyline of cadastral data set with the nearest polylines of a city planning map are presented.

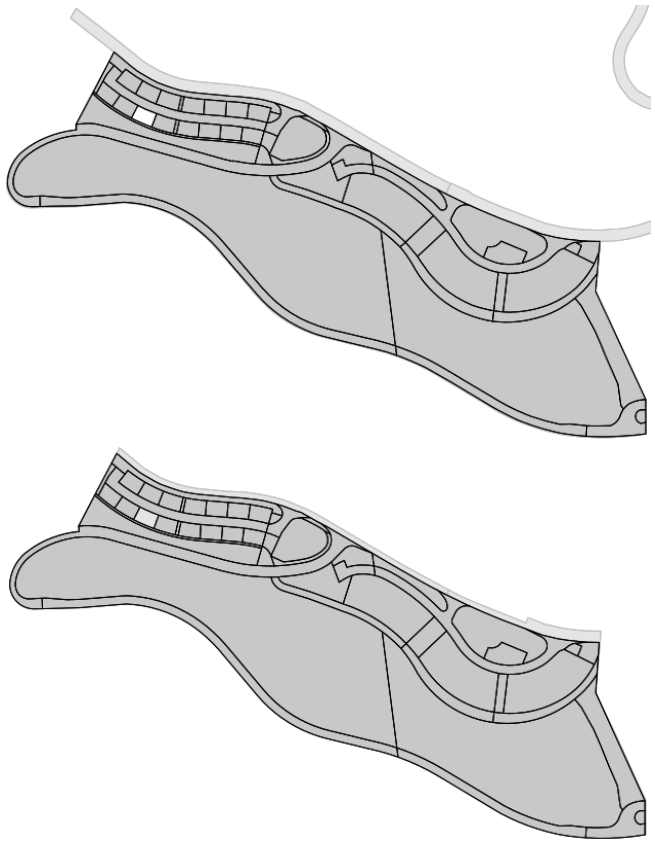

Figure 8. The result of triangulation. Upper – cadaster, lower – city planning. Nesher.

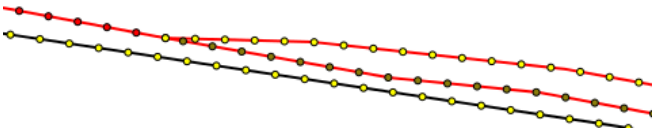

Figure 9. Equidistant points used to calculate similarity of polylines and polylines' segments. Red line – city planning dataset, black – cadastre.

The algorithm for line pairs searching is presented in pseudo code in Figure 10.

The pseudo-function gets the 'id's_of_closest_lines () and returns a pair of neighboring lines' ids points which are closest. Usually, for one line A, several pairs of ids can be

defined (idA1-idB1, idA1-idB2, …). All id pairs are processed. Pts_A – points of a city planning dataset are situated on a line with id idA; Pts_B – points of line idB (cadastral map). The pseudo function gets_segments (Pts_A, Pts_B) and splits lines into segments at intervals where the distance between the nearest points is more than D_{max}. In the first line of the pseudo function - finding_pairs (PtsA,PtsB) - we test distances from start point of line A to start and end points of line B. If start-start distance is more than start-end, we invert the order of points in line A. Then we set l,i,j variables: l – length of line, i - number of starting points on line A, j - number of starting point on line B.

```
Foreach idA,idB in get_ids_of_closest_lines(){
   Pts_A    = get_points('city planning',idA)
   Pts_B =    get_points('cadastre',idB)
   If min(len(Pts_A),len(Pts_B)) > tail {
     Foreach segm in get_segments(Pts_A,Pts_B){
        Pts_A_segm=segm['Pts_A']
        Pts_B_segm=segm[' Pts_B']
        Result_line_pair=find_pair(Pts_A_segm,Pts_B_segm)}}}

Function find_pair(PtsA,PtsB) {
  If (distance(PtsA[0],PtsB[0]) >
   distance(PtsA[0], PtsB[-1])){ PtsA=reverse(PtsA)  }
  Length=min(len(PtsA), len(PtsB))
  Global_measures=[ ]
  Foreach l in reverse([tail,…,length]){
   Local_measures=[ ]
   Foreach i in [0,…,len(PtsA)-tail]{
    Foreach j in [0,…,len(PtsB)-tail]{
      cur_measure=Calc_measures(PtsA,PtsB,i,j,l)
      if (cur_measure[0]<max_stand_dev and
          cur_measure[1]<max_distance){
        Local_measures.append(cur_measure)} } }
   Global_measures
.append(Find_local_indicator(Local_measures))
   If    Global_measures    and    (l==length    or
len(Global_measures)>tail){
     Gen_desc_list=[calculate_global_indicator(cur) for cur in
                    Global_measures]
     If max(Gen_desc_list[:-tail])> max(Gen_desc_list[-tail:]){
       Return
Global_measures[index_of_maximal(Gen_desc_list)]}}}

Function Calc_measures(PtsA,PtsB,i,j,l){

  cur_PtsA= PtsA[i:i+l]
  cur_PtsB= PtsB[j:j+l]
  dists=Distances(cur_PtsA,cur_PtsB)

  Return [stand_dev(dists),max(dists),
   min(dists),delta_x,delta_y,
   get_max_stddev_of_tailes(cur_PtsA,cur_PtsB),
   get_min_shape_descr_of_tails(cur_PtsA,cur_PtsB),
   get_shape_descriptor(cur_PtsA,cur_PtsB),   i, j, l]}
```

Figure 10. Searching for equal polylines or polylines' segments.

The function Calc_measures (PtsA, PtsB, i, j, l) and calculates a set of parameters (standard deviation of distances, shape descriptor, minimal shape descriptor of line tails, minimal and maximal distance between points). This enables us to define similarity of line A segment from i to i+l and for line B - from j to j+l. Variables i and j, which define the optimal segment (pseudo function Find_local_optimal (Local_measures)), have been found for each possible length l using (2).

$$Loc\_Ind = \frac{(d - D_{\max})}{-D_{\max}} + \frac{s - S_{t\_\max}}{S_{t\_\max} - S_{t\_\min}} \quad (2)$$

$$G\_Ind = \frac{\sigma_t - \sigma_{\max}}{\sigma_{\max}} + \frac{d - D_{\max}}{-D_{\max}} + \frac{s - S_{t\_\max}}{S_{t\_\max} - S_{t\_\min}} + (1 - l)/\mathrm{L} \quad (3)$$

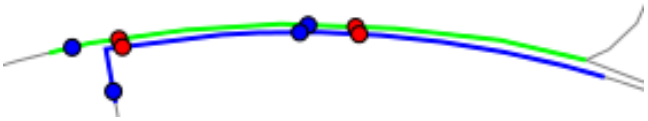

Figure 11. Segment of line A (city planning) – green; segment of line B (cadastre) – blue. Start and end point of the most similar line segments are red points (i=6,j=6, Loc_ind=0.86); blue points – i=2, j=1, Loc_Ind=0.026.

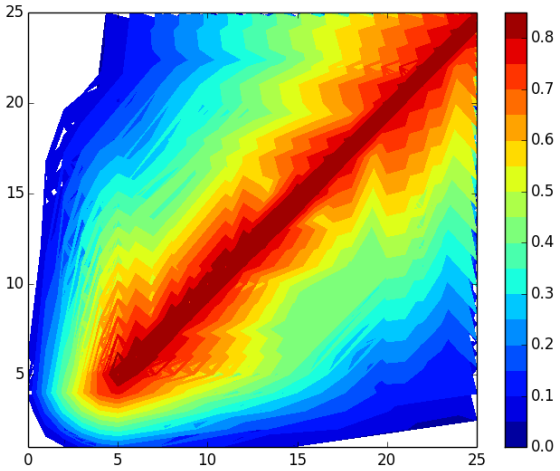

Figure 12. Plot of indicator Loc_Ind: X axis – i, Y axis - j. The segment with i=6 and j=6 is the most optimal.

The meaning of parameters in (2): d – maximal distance between points of lines A and B for (l,i,j), s - minimal tail shape descriptor. In this step, we have a Global_measures list containing elements that correspond to some l and contain

measures of line segments with maximal indicator Loc_Ind derived from the list (Local_measures) with variable (i,j).

This process is illustrated in Figures 11 and 12 (segment length is 20 meters). Figure 11 depicts different segments with the same length; Figure 12 is a plot of indicator Loc_Ind depicted by color. The next stage is defining optimal segment length. In the previous stage, we defined optimal segments i,j for some length l by calculating local indicator Loc_Ind. To define optimal segment length we use global indicator G_Ind; its formula is presented as (3).

In the equation, σ_t means maximal standard deviation of point distances of line segments' tails; for more details see Section III and (2). The resulting optimal line length is defined by maximal global indicator G_Ind. The process is illustrated in Figures 13 and 14. Figure 13 depicts examples of optimal segments with different lengths. Figure 14 is a plot of indicator G_Ind. It is obvious that the optimal segment length is 41 meters (element with maximal G_Ind, according to the plot presented in Figure 14).

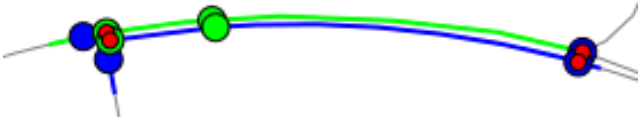

Figure 13. Segment of line A (city planning) – green; segment of line B (cadastre) – blue. Nodes of the most similar line segments with different lengths of segment: red points – l=41,i=5,j=5,G_Ind=2.35; green points – l=10,i=5,j=5 G_Ind=1.69; blue points – l=43,i=3,j=3 G_Ind=1.64.

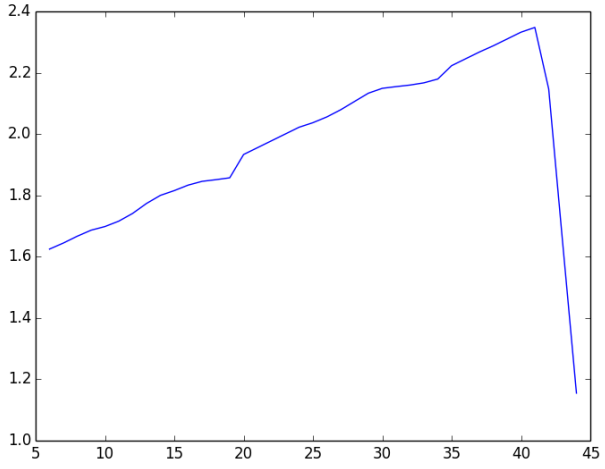

Figure 14. Plot of indicator G_ind: X axis – segment length (in meters), Y axis – G_Ind.

VI. COMPILING A FINAL MAP

At this point, we have the pairs of corresponding polygons and polygon boundary segments. Some segments are overlapped; to resolve conflicts, a special parameter was developed:

$$P = \frac{l - \min\_len}{range\_len + \frac{\sigma}{-\sigma_{\max}}} \quad (4)$$

where: l is length of line of one of the lines in a line pair, min_len – minimal length of line of all line pairs, range_len – range of length of all line pairs. A line pair with maximal P will be saved; others will be removed. The process is shown in Figure 15.

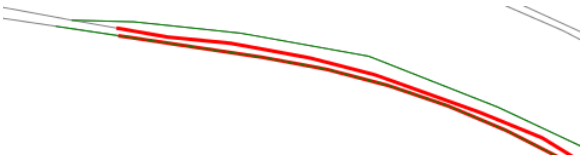

Figure 15. Overlaped line pairs: red line pair – P=1.23, green line pair – P=1.09 . Green line pair will be removed.

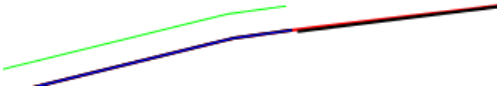

Figure 16. Moving segments without pairs and closing boundaries: green – lines that do not have pairs in cadastral dataset, blue – moved green lines, red – closing boundary by moving nodes, black – cadastral pair of city-planning line segments.

After removing overlapping line pairs, we can use a correspondent line segment of the cadastral dataset instead of the city-planning dataset. The boundaries of counterpart polygons are extracted from polygon maps calculated by triangulation. The pair segments are composed with the extracted boundaries. We will use nodes of pair segments and centroids of pair polygons as control ground points for transformation.

The lines and line segments of the city-planning dataset without corresponding lines of the cadastral dataset have been moved. Delta X and delta Y have been calculated as average delta X and delta Y of neighboring nodes of line pairs and centroids of pair polygons. Unclosed boundaries of polygons have been closed by moving the nodes of an unclosed line to the nearest node of a neighboring line (see Figure 16).

We now have a map containing closed boundaries. All legacy centroids have been removed. To create polygons we add a new centroid to each set of closed boundaries (see Figure 17). As mentioned above, we have prepared a list of control ground points. The list contains coordinates of correspondent polygons' centroids and line pairs' nodes. We use these control ground points to transform the original city planning dataset to its accurate position. The transformed city panning dataset could be used as a product by itself, because it is a more accurate dataset in comparison to the original map. But we will use it mainly for detecting attributes of the resulting dataset.

As mentioned earlier, the legacy centroids have been removed from the resulting map and new centroids have been added. In other words, we have removed all connections between result and original datasets. To define the final correspondences we need to apply the triangulation process one more time, but now we will compare the results (Figure 17) with the transformed city planning dataset. For each polygon a correspondence must be found, otherwise the polygon will be marked as an error polygon. We have applied the triangulation to both datasets. For each polygon a correspondence has been established; no error polygons have been detected.

Figure 17. Adding centroids to closed boundaries. Yokne'am.

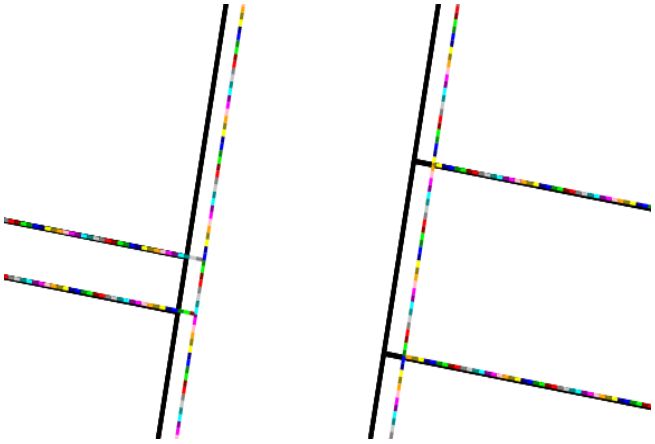

Figure 18. Calculation of minimal distances from segmented boundaries of the result dataset (color segments) to cadastre dataset (black lines).

VII. RESULTS

In order to evaluate the results, we use distances between the boundaries of the result and cadastre datasets as a main measure. As mentioned in the previous section, no error polygons have been defined by the triangulation, i.e., no semantic errors have been detected.

In order to evaluate the geometrical accuracy of the resulting map, the result boundaries have been split to 0.5 m segments and a minimal distance to the boundaries of the cadastre dataset is calculated. In Figure 18 the prepared color segments and black lines are presented. The minimal distance from each segment to the closest point on the nearest line is calculated.

TABLE I. GEOMETRIC ACCURACY OF THE RESULT DATASETS

Measures in meter	**Nesher datasets compared with cadastre boundaries**		
	Original	***Transformed***	***Result***
Mean distance	0.59	0.55	0.01
Standard deviation	0.55	0.54	0.05
	Yokne'am datasets compared with cadastre boundaries		
Mean distance	0.82	0.63	0.13
Standard deviation	0.81	0.84	0.77

Table I presents the geometric accuracy of the result datasets estimated by the distances between result boundary segments and cadastre (accurate) dataset. We can conclude that positional accuracy has been significantly improved for the result datasets. The accuracy of transformed maps has only slightly improved.

In addition to the table, the histograms of distances are presented in Figures 19 and 20. The vertical axis of the histogram is the number of segments, the horizontal axis depicts distance in meters. The histograms also prove the significant improvement in positional accuracy.

In contrast to the mean and the standard deviation values presented in the table, we cannot unambiguously conclude from the histograms in Figure 19 (Nesher dataset), that the transformed map is more accurate. The main reason for this is the fact that most polygons of the map have correspondences, and only several northern polygons do not. That leads to a situation of the presence of one type of control ground points (line pair nodes) in the northern part of the dataset only, with another type in the remaining area. For larger and more differentiate datasets this would not work. The histogram of result datasets depicts significant improvement of positional accuracy in comparison to original and transformed maps.

As mentioned earlier, in the case of a larger and more differentiate dataset, the contrast between the histograms of original and transformed datasets will be clearer. We can see this in Figure 19.

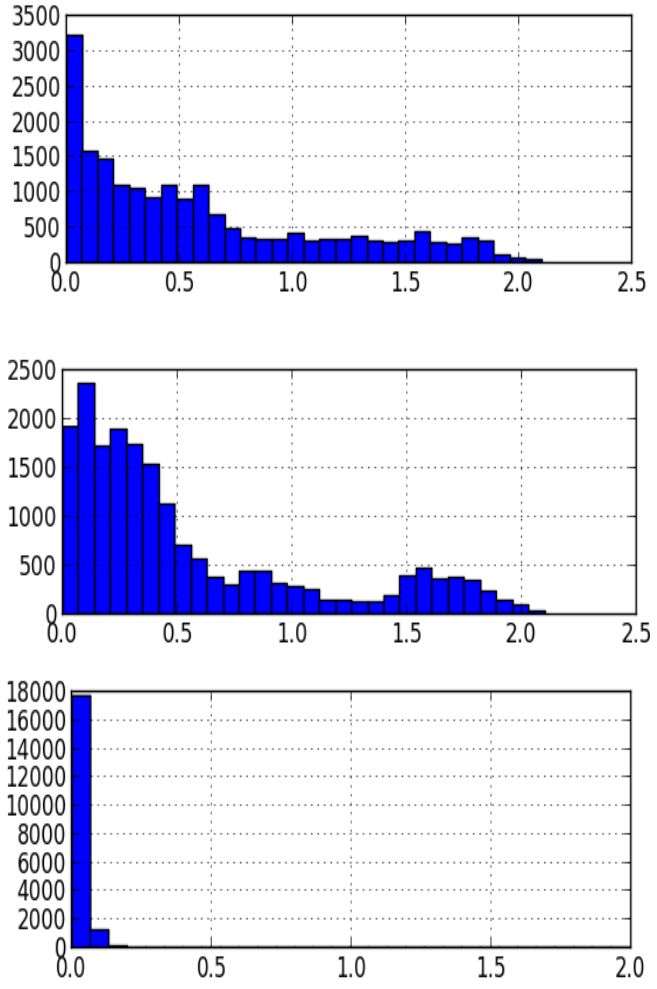

Figure 19. Histogram of distances. From top to bottom: original, transformed, and result datasets. Nesher. X axis – distance in meter, Y axis – number of segments.

Results are presented in Figure 21 for six extents. The extents correspond to the red numbers in circles in Figure 1. Color background is the result dataset. It is overlaid by red and black lines. Red lines are the original city planning dataset. Black lines are the cadastre dataset. We can conclude that most line segments have been taken from the cadastral dataset; others have been transformed to correspond with cadastral polyline segments. The result looks satisfactory; the final map is holistic and does not contain significant deficiencies. A review implemented by specialists enables us to state that the results are satisfactory and the approach could be used in real applications after fixing some lacks.

VIII. CONCLUSION

An approach for improving the accuracy of polygons' data is presented. Land-use city planning dataset locations have been corrected according to the cadastral dataset. The polylines' segments along the polygons have been split by equidistant points. Analysis has been performed using statistics based on the points of the neighboring polylines of the two datasets. A set of parameters has been used: shape descriptor of polyline segments, standard deviation of point distances, minimal and maximal point distances, standard deviation of segment tails, etc. A set of correspondent polyline segments using special indicators has been found. It enables us to find optimal segments from the list of polyline segments with different lengths and starting points.

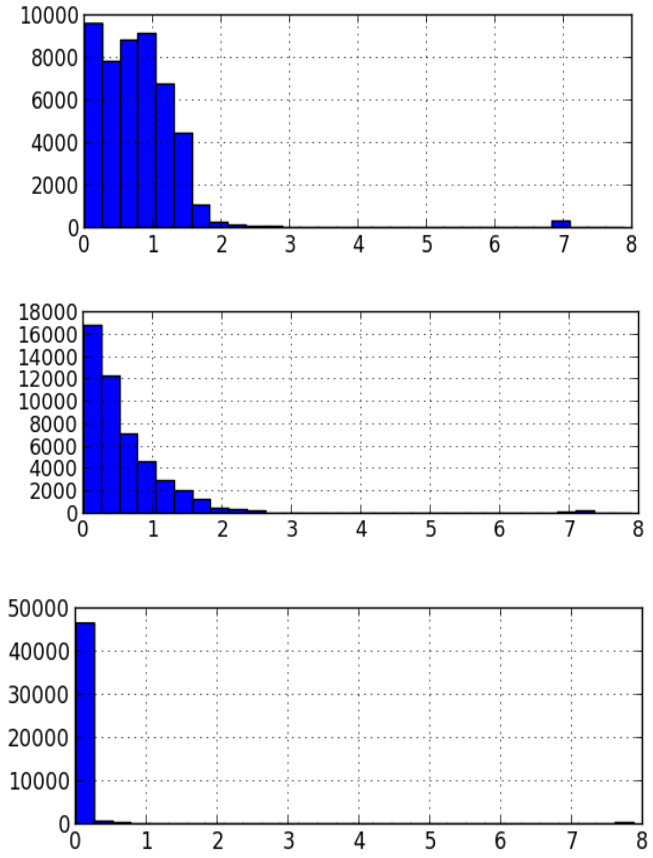

Figure 20. Histogram of distances. From top to bottom: original, transformed, and result datasets. Yokne'am. X axis – distance in meter, Y axis – number of segments.

The polyline segments of the city planning data with parameters similar/identical to the segments of the cadastral data were linked to these segments (defining counterpart segments). Segments without a counterpart were transformed. The triangulation process has been used to define correspondences between polygons. It enables us to find optimal segments from the list of polyline segments with different lengths and starting points.

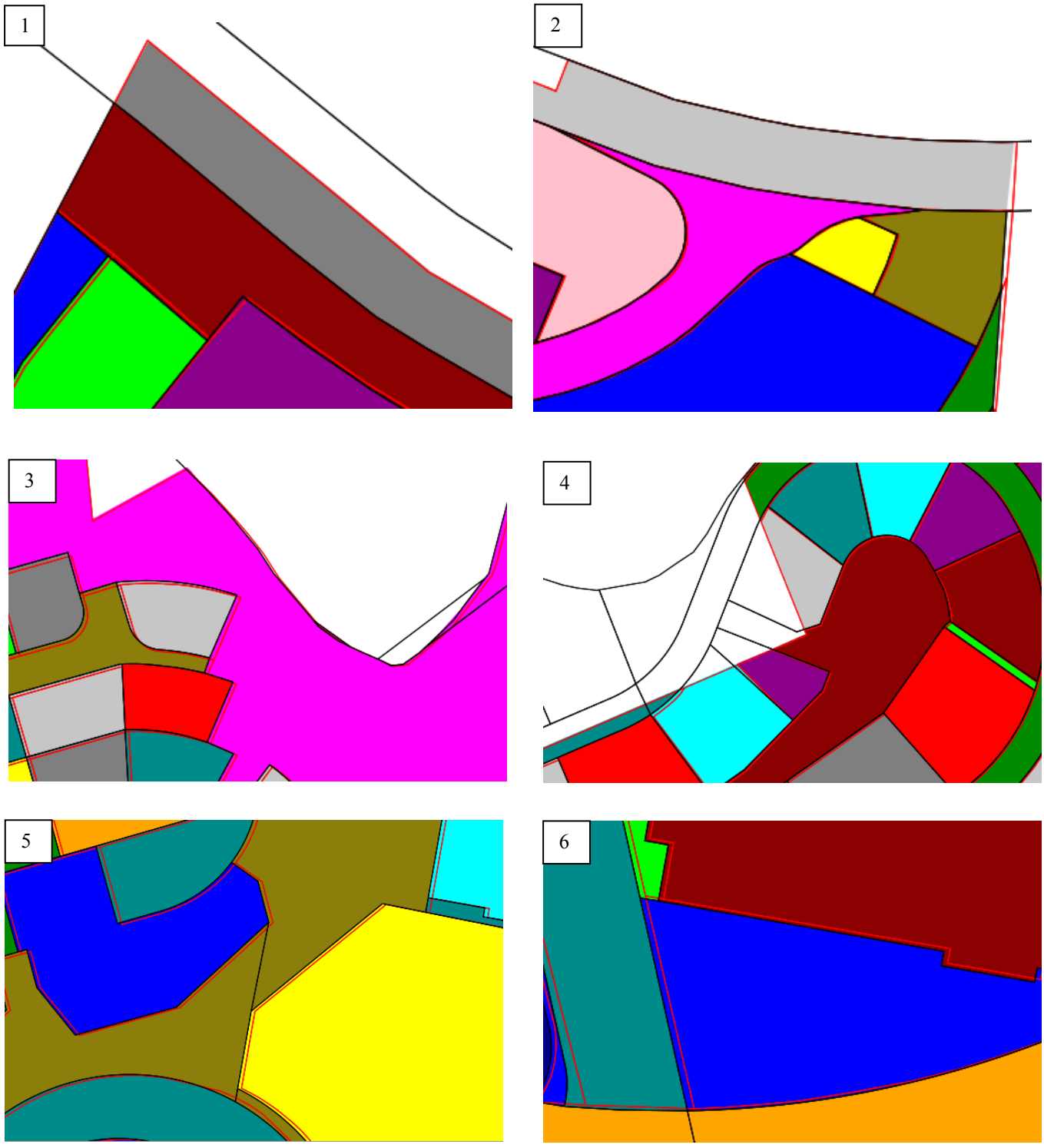

Figure 21. Results. Color background is result polygons. Red lines are original city planning polygons' boundaries. Black lines are cadastre polygons' boundaries. 1 and 2 - Nesher; 3-6 – Yokne'am. Figure 1 depicts the positions of the extents.

The polyline segments of the city planning data with parameters similar/identical to the segments of the cadastral data were linked to these segments (defining counterpart segments). Segments without a counterpart were transformed. The triangulation process has been used to define correspondences between polygons.

In the future, we need to test the approach with additional datasets and different parameters, to compare it with other approaches, and to improve calculation speed.

To implement the approach, we used Python 2.7 programming language (with numpy, scipy and matplotlib additional libraries), GRASS GIS 7.1, and Debian GNU/Linux 7 operating system.

ACKNOWLEDGEMENT

This research was supported by the Survey of Israel as a part of Project 2019317. The authors would like to thank the Survey of Israel for providing financial support and data for the purpose of this research.

REFERENCES

[1] A. Noskov and Y. Doytsher, “A Segmentation-based Approach for Improving the Accuracy of Polygon Data,” GEOProcessing 2015, 2009, Portugal, pp. 69-74.

[2] S. Filin and Y. Doytsher, “The detection of corresponding objects in a linear-based map conflation,” Surveying and land information systems, vol. 60(2), 2000, pp. 117-127.

[3] V. Walter and D. Fritsch, “Matching spatial data sets: a statistical approach,” International Journal of Geographical Information Science (IJGIS), vol. 13 (5), 1999, pp. 445–473.

[4] X. Shu and X. Wu. “A novel contour descriptor for 2D shape matching and its application to image retrieval”, Image and vision Computing, vol. 29.4, 2011, pp. 286-294.

[5] S. Belongie, J. Malik, and J. Puzicha, “Shape matching and object recognition using shape contexts,” IEEE Trans. on Pattern Analysis and Machine Intelligence, 24(4), 2002, pp. 509-522.

[6] E. Safra, Y. Kanza, Y. Sagiv, C. Beeri, and Y. Doytsher, “Ad-hoc matching of vectorial road networks,” International Journal of Geographical Information Science, iFirst, 2012, pp. 1–40, ISSN: 1365-8816, ISSN: 1362-3087.

[7] B. Schmitzer and S. Christoph, "Object segmentation by shape matching with Wasserstein modes," Energy Minimization Methods in Computer Vision and Pattern Recognition, Springer Berlin Heidelberg, 2013.

[8] T. Ma and J. Longin, "From partial shape matching through local deformation to robust global shape similarity for object detection," Computer Vision and Pattern Recognition (CVPR), IEEE Conference on. IEEE, 2011, pp. 1441-1448.

[9] X. Chen, “Spatial relation between uncertain sets,” International archives of Photogrammetry and remote sensing, vol. 31(B3), Vienna, 1996, pp. 105-110.

[10] A. Bronstein, R. Kimmel, M. Mahmoudi, and G. Sapiro, “A Gromov-Hausdorff framework with diffusion geometry for topologically-robust non-rigid shape matching,” International Journal of Computer Vision, vol. 89(2-3), 2010, pp. 266-286.

[11] P. Tahmasebi, A. Hezarkhani, and M. Sahimi, “Linking Objects of Different Spatial Data Sets by Integration and Aggregation,” vol. 2(4), 1998, pp. 335-358.

[12] E. Rahm and P. Bernstein, “A survey of approaches to automatic schema matching,” The International Journal on Very Large Data Bases (VLDB), vol. 10(4), 2001, pp. 334–350.

[13] V. Kashyap and A. Sheth, “Semantic and schematic similarities between database objects: a context-based approach,” The International Journal on Very Large Data Bases (VLDB), vol. 5(4), 1996, pp. 276–304.

[14] P. Shvaiko and J. Euzenat, “A survey of schema-based matching approaches,” Journal on Data Semantics IV, Springer Berlin Heidelberg, 2005, pp. 146-171.

[15] B. Amann, C. Beeri., I. Fundulaki, and M. Scholl., “Ontology-based integration of XML Web resources,” 1st International Semantic Web Conference (ISWC), Sardinia, Italy, June 9-12 2002, pp. 117–131.

[16] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, “Approximate String Joins in a Database (Almost) for Free,” Proceedings of the 27th International Conference on Very Large Data Bases, Italy, 2001.

[17] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins in an RDBMS for web data integration," Proceedings of the 12th international conference on World Wide Web. ACM, 2003.

[18] C. Parent and S. Spaccapietra, “Database integration: the key to data interoperability,” Advances in Object-Oriented Data Modeling, M. P. Papazoglou, S. Spaccapietra, Z. Tari (Eds.), The MIT Press, 2000.

[19] G. Wiederhold, “Mediation to deal with heterogeneous data sources,” Interoperating Geographic Information System, 1999, pp. 1–16.

[20] A. Saalfeld, “Conflation-automated map compilation,” International Journal of Geographical Information Science (IJGIS), vol. 2 (3), 1988, pp. 217–228.

[21] M. Sester, K. Anders, and V. Walter, “Linking Objects of Different Spatial Data Sets by Integration and Aggregation,” GeoInformatica, vol. 2(4), 1998, pp. 335-358.

[22] C. Steger, M. Ulrich, and C. Wiedemann, “Machine vision algorithms and applications”, Weinheim: wiley-VCH, 2008, pp. 1-2.

[23] G. Bradski and A. Kaehler, “Learning OpenCV: Computer vision with the OpenCV library,” O'Reilly Media, Inc, 2008.

[24] J. Howse, “OpenCV Computer Vision with Python,” Packt Publishing Ltd, 2013, ISBN: 978-1-78216-392-3.

[25] J. Shewchuk, “Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator,” Applied computational geometry towards geometric engineering, Springer Berlin Heidelberg, 1996, pp. 203-222.

[26] F. Aurenhammer, “Voronoi diagrams - a survey of a fundamental geometric data structure,” ACM Computing Surveys (CSUR), vol. 23(3), 1991, pp. 345-405.

[27] M. Landa, “GRASS GIS 7.0: Interoperability improvements,” GIS Ostrava, Jan. 2013, pp. 21-23.

[28] R. Blazek, M. Neteler, and R. Micarelli, "The new GRASS 5.1 vector architecture," Open source GIS-GRASS users conference, University of Trento, Italy, 2002.

[29] J. Herring, "OpenGIS Implementation Standard for Geographic information-Simple feature access-Part 1: Common architecture," OGC Document 4, no. 21, 2011.

[30] I. Wilson, J.M. Ware, and J.A. Ware, “A genetic algorithm approach to cartographic map generalisation” Computers in Industry, vol. 52(3), 2003, pp. 291-304.

[31] B. Gaster, L. Howes, D. Kaeli, P. Mistry, and D. Schaa, “Heterogeneous Computing with OpenCL: Revised OpenCL1,” Newnes, 2012.