Calculating the parameters of 2D Helmert transformation by robust method

G. Nagy*
*Óbuda University, Alba Regia Technical Faculty, Institute of Geoinformatics
nagy.gabor@amk.uni-obuda.hu

***Abstract*—Helmert transformation is an essential tool of the geodetic calculations. In most cases we use less squares method to calculate the parameters of the transformation, but this way is sensitive when the control points contains outlier data, for example a point has wrong coordinates. This paper describes a robust method to calculate the parameters of Helmert transformation, and analyze the result, compare with the less squares method.**

***Index Terms*—Helmert transformation, coordinate transformation, 2D transformation**

I. Introduction

Many coordinate systems are used in the geodesy practice, these systems may be standardized grids or local coordinate systems. When we would like to make a connection between two coordinate system, we need functions which provides the coordinates from the coordinates of the another system.

There are many functions for this task, one of the simplest is the linear connection, which is called affine transformation. The equation of the affine transformation, from system A to system B:

$$\begin{aligned} x_1^B &= t_1^{AB} + l_{11}^{AB} x_1^A + l_{12}^{AB} x_2^A \\ x_2^B &= t_2^{AB} + l_{21}^{AB} x_1^A + l_{22}^{AB} x_2^A \end{aligned} \tag{1}$$

The equations can be written by vectors and matrices:

$$\mathbf{x}_B = \mathbf{t}_{AB} + \mathbf{L}_{AB}\mathbf{x}_A \tag{2}$$

Where $\mathbf{t}$ is the vector of translation, and $\mathbf{L}$ is an arbitrary matrix ($|\mathbf{L}| \neq 0$) of the linear transformation.

The coordinate transformation can be calculated by one matrix multiplication:

$$\begin{bmatrix} x_1^B \\ x_2^B \\ 1 \end{bmatrix} = \begin{bmatrix} l_{11}^{AB} & l_{12}^{AB} & t_1^{AB} \\ l_{21}^{AB} & l_{22}^{AB} & t_2^{AB} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1^A \\ x_2^A \\ 1 \end{bmatrix} \tag{3}$$

A. Helmert transformation

The Helmert transformation is a special case of the affine transformation, when $\mathbf{L} = \mu\mathbf{R}$, where $\mathbf{R}$ is a rotation matrix, and μ is a scale factor. In this case $l_{11} = l_{22} = a = \mu\cos\alpha$ and $l_{12} = -l_{21} = b = \mu\sin\alpha$, where a and b is parameters of the Helmert transformation (with the vector of translation), and α is the rotation angle.

The Helmert transformation by one matrix multiplication:

$$\begin{bmatrix} x_1^B \\ x_2^B \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_1^{AB} \\ -b & a & t_2^{AB} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1^A \\ x_2^A \\ 1 \end{bmatrix} \tag{4}$$

The scale factor and the rotation angle can be calculated from a and b linear parameters:

$$\begin{aligned} \mu &= \sqrt{a^2 + b^2} \\ \alpha &= \arccos\left(\tfrac{a}{\mu}\right) = \arcsin\left(\tfrac{b}{\mu}\right) = atan2\,(b, a) \end{aligned} \tag{5}$$

The coordinates and the parameters may be handled as complex numbers:

$$\left(x_1^B + x_2^B i\right) = \left(t_1^{AB} + t_2^{AB} i\right) + \left(x_1^A + x_2^A i\right) \cdot (a - bi) \tag{6}$$

or shortly:

$$\mathbf{x}^B = \mathbf{t}^{AB} + (a - bi)\,\mathbf{x}^A \tag{7}$$

If we have $\mathbf{c}$ and $\mathbf{d}$ points in A and B coordinate system, the equations are:

$$\mathbf{c}^B = \mathbf{t}^{AB} + (a - bi)\,\mathbf{c}^A \tag{8}$$

$$\mathbf{d}^B = \mathbf{t}^{AB} + (a - bi)\,\mathbf{d}^A \tag{9}$$

The $\mathbf{t}^{AB}$ parameter can be eliminate, if we subtract (9) from (8):

$$\mathbf{d}^B - \mathbf{c}^B = \left(\mathbf{d}^A - \mathbf{c}^A\right) \cdot (a - bi)$$

and the a and b parameters of the Helmert transformation can be calculated by a complex expression:

$$(a - bi) = \frac{\mathbf{d}^B - \mathbf{c}^B}{\mathbf{d}^A - \mathbf{c}^A} \tag{10}$$

B. Calculate parameters

The affine transformation has 6 (two elements of the vector of the translation, and four elements of the matrix of the linear transformation), and the Helmert transformation has 4 parameters (two elements of the vector of the translation, and two independent values – a and b – of the matrix of the linear transformation). These parameters can be calculated if we know some points coordinates in both system, which are

called control points in the following. The equations of (1) may be elements of a system of linear equations in each point, where the variables are the parameters of the transformation. (The coordinates are known in both systems.) Each control point provides two equations.

The affine transformation needs minimum 3 control points, because these three control points provides $3 \cdot 2 = 6$ equations for the 6 parameters of the transformation. The Helmert transformation needs minimum 2 control points, because these two control points provides $2 \cdot 2 = 4$ equations for the 4 parameters of the transformation. If we have more point than the minimum requirement, the system of equations will be over-determined. The most popular method for solving these over-determined systems of equations is the least square method. The least squares method determines the parameters of the transformation so, that the sum of the squares of the errors (the difference between the control point position in the system B and the transformed position from the system A) will be the least.

C. Linear regression by robust methods

Least squares method provides the most likely result, if the errors of the measurements have normal distribution. In practice, the measurements may have outliers, where the error is larger than the expected. We usually qualify a measurement as an outlier if the error is larger than the triple of the standard deviation. The outlier measurements make an bad influence to the result of the least squares method.

The robust methods can calculate good result with outliers. For example, the impact of a wrong measurement is not too remarkable in the median, but the mean may be wronger. There is an set of numbers: 1.012, 1.008, 10.25, 1.002, 0.9998, 10.007, 0.9993, 0.9988, 1.003, 0.9990. The mean of this data set is 2.83419, but the median is 1.0025. The mean without the outlier is 1.0027375 (the median is 1.001). The outlier values (10.25, 10.007) make wrong the mean result, but the median produces a value around the most of the dataset numbers (except the outliers).

There are many robust method for calculating a linear regression. [1], [2] For example RANSAC (Random Sample Consensus) [3], [4], [5] or the Theil-Sen estimator [6]. The SBLR [7] is also such method, which is a multidimensional extension of the Fitting Disc Method [8]. These methods provide a good regression when the dataset contains outlier points.

Any linear regression method may be suitable for calculating the parameters of the affine transformation. The explanatory variables are the coordinates of the system A, and the dependent variables are the coordinates of the system B; both coordinates have a separately linear regression.

II. A robust method for 2D Helmert transformation

The 2D Helmert transformation has four parameters: two translation, one rotation (the angle of the 2D rotation, denoted α) and one scale factor (denoted μ). Instead of α and μ, the equation (4) uses a and b, the element of the matrix of the linear transformation.

The median is a simple and robust method on the one dimensional datasets, but it is not work in multiple dimensions. This paper describes a method, that divides the calculation more steps, and in these steps uses median calculation.

We can calculate the rotation angle (α) and the scale factor (μ) from two control points, denoted $\mathbf{c}$ and $\mathbf{d}$. The first step, the a and b can be calculated as an complex equation (see (10)):

$$a - bi = \frac{\left(d_1^B - c_1^B\right) + \left(d_2^B - c_2^B\right) i}{\left(d_1^A - c_1^A\right) + \left(d_2^A - c_2^A\right) i} \tag{11}$$

The rotation angle (α) and the scale factor (μ) can be calculated by the equation (5). If we have n control points, we can make $\frac{n(n-1)}{2}$ point pairs, and each point pair provides an α and a μ parameters. The final parameters are calculated by weighted median [9]; the weights are the distance between the two control points, if this is less than the median of these distances, or the median distance, if the distance is larger that the median distance.

After we have the final α and μ values (and the $\mathbf{L}$ matrix with a and b), we can calculate the element of the translation vector, by this equation derived from (2):

$$\mathbf{t}_{AB} = \mathbf{x}_B - \mathbf{L}_{AB}\mathbf{x}_A \tag{12}$$

The translation vector can be calculated with each control point. The final values of the elements of the translation vector will be the median of these values.

III. Numerical tests

A. Implementation

I have written a Python module called **helmert** for calculating the parameters of a Helmert transformation from control points by the least square method and the recommended robust method. The source code of this module is available in the Appendix A.

This Python module can be used simply from other Python modules:

```
from helmert import helmert_lsq,helmert_rob
from helmert import trp_helmert

#Helmert transformation
#from A to B coordinate system

#Points in A and B coordsys
ptA=[(1.0,1.0),(2.0,1.0),(1.0,2.0),(2.0,2.0)]
ptB=[(2.5,3.7),(3.3,3.1),(3.1,4.5),(3.9,3.9)]

#Test points in A coordsys
pt2A=[(1.2,1.8),(1.5,1.7)]

#Calculate the parameters of the transformation
tr_lsq=helmert_lsq(ptA, ptB)
tr_rob=helmert_rob(ptA, ptB)
```

```
#Calculate the transformated coordinates
#by different transformation parameters
pt2B_lsq=trp_helmert(pt2A, tr_lsq)
pt2B_rob=trp_helmert(pt2A, tr_rob)
```

The **trp_helmert** function returns the transformed coordinates by an Helmert transformation. The input is in the first (**ptl**) argument, the output is the return value of the function. The coordinates are in (x, y) tuples, and the parameters are in a (dx, dy, a, b) tuple in the second (**trp**) argument.

The **helmert_lsq** and **helmert_rob** functions calculate the parameters of an Helmert transformation from points which are known in both coordinate system. The points are in lists of tuples, the first argument is the points of the initial (A), and the second argument is the points of the aim (B) coordinate system. The return values are the parameters of the Helmert transformation in a tuple (same as in the **trp_helmert** function). The **helmert_lsq** function uses the classical less squares method, and the **helmert_rob** function uses the recommended robust method.

In the following studies I use this module for testing the recommended method.

B. Studying the result

I study the result of the method with random datasets. The initial position is a random place with coordinate between 0 and 1000. The initial transformation is also random with a translation vector coordinates between -1000 and 1000, the μ between 0.999 and 1.001 and the α between $-\pi$ and π. The coordinates of the second system are calculated by the initial transformation, and add a random normal distribution error with 0.1 standard deviation by the **normalvariate** function of the **random** Python module. The outliers has an additional error between 10 and 50 by the **uniform** function of the **random** module.

A program create one million datasets in each studied cases. The number of control points are between 5 and 19, and the number of outliers are between 0 and the third of the control points. The random datasets are generated by the function of **testtool.py** module, whose source code is in the Appendix B.

The transformation was calculated in each dataset, and the next step the test program calculates the mean squared error between the original (without any error) and the transformed positions:

$$err = \sqrt{\frac{\sum \left(\left(x_1^{TR} - x_1^{OR}\right)^2 + \left(x_2^{TR} - x_2^{OR}\right)^2 \right)}{n}} \qquad (13)$$

The outliers are excluded from the sum. The points which are in the calculation of (13) contains only normal distribution error with $\sigma = 0.1$.

The test program counts the number of the datasets where the mean squared error is less than the single, the double, the triple, etc. of the standard deviation of the random error of the points ($\sigma = 0.1$).

Table I
THE RATES OF MEAN ERRORS IN n POINT DATASET WITH o OUTLIERS

n	o	$< \sigma$	$< 2\sigma$	$< 3\sigma$	$< 4\sigma$	$< 5\sigma$
5	0	78.601%	99.997%	>99.999%	>99.999%	>99.999%
5	1	51.596%	85.872%	89.016%	89.662%	89.884%
6	0	78.674%	99.999%	>99.999%	>99.999%	>99.999%
6	1	61.809%	98.072%	99.118%	99.296%	99.353%
6	2	27.657%	50.261%	53.211%	54.048%	54.488%
7	0	74.721%	>99.999%	>99.999%	>99.999%	>99.999%
7	1	62.930%	99.850%	99.989%	99.994%	99.996%
7	2	37.275%	73.938%	77.385%	78.223%	78.603%
8	0	74.949%	>99.999%	>99.999%	>99.999%	>99.999%
8	1	66.425%	99.993%	>99.999%	>99.999%	>99.999%
8	2	46.827%	90.325%	92.605%	93.078%	93.274%
9	0	72.142%	>99.999%	>99.999%	>99.999%	>99.999%
9	1	65.211%	99.998%	>99.999%	>99.999%	>99.999%
9	2	50.987%	97.472%	98.489%	98.651%	98.705%
9	3	31.845%	72.453%	76.124%	77.033%	77.482%
10	0	72.281%	>99.999%	>99.999%	>99.999%	>99.999%
10	1	66.854%	>99.999%	>99.999%	>99.999%	>99.999%
10	2	55.880%	99.569%	99.813%	99.845%	99.856%
10	3	39.433%	86.594%	89.224%	89.827%	90.095%
11	0	70.083%	>99.999%	>99.999%	>99.999%	>99.999%
11	1	65.536%	>99.999%	>99.999%	>99.999%	>99.999%
11	2	57.036%	99.945%	99.989%	99.993%	99.994%
11	3	43.625%	94.583%	96.189%	96.510%	96.642%
12	0	70.360%	>99.999%	>99.999%	>99.999%	>99.999%
12	1	66.396%	>99.999%	>99.999%	>99.999%	>99.999%
12	2	59.761%	99.997%	>99.999%	>99.999%	>99.999%
12	3	48.747%	98.332%	99.016%	99.131%	99.172%
12	4	35.243%	85.370%	88.190%	88.852%	89.169%
13	0	68.657%	>99.999%	>99.999%	>99.999%	>99.999%
13	1	65.120%	>99.999%	>99.999%	>99.999%	>99.999%
13	2	59.501%	>99.999%	>99.999%	>99.999%	>99.999%
13	3	50.667%	99.591%	99.814%	99.843%	99.855%
13	4	39.291%	92.986%	94.886%	95.276%	95.454%
14	0	68.887%	>99.999%	>99.999%	>99.999%	>99.999%
14	1	65.877%	>99.999%	>99.999%	>99.999%	>99.999%
14	2	61.234%	>99.999%	>99.999%	>99.999%	>99.999%
14	3	53.887%	99.931%	99.975%	99.980%	99.982%
14	4	43.993%	97.179%	98.145%	98.325%	98.404%
15	0	67.458%	>99.999%	>99.999%	>99.999%	>99.999%
15	1	64.673%	>99.999%	>99.999%	>99.999%	>99.999%
15	2	60.578%	>99.999%	>99.999%	>99.999%	>99.999%
15	3	54.478%	99.990%	99.998%	99.999%	99.999%
15	4	46.049%	99.025%	99.461%	99.529%	99.556%
15	5	36.144%	92.205%	94.195%	94.622%	94.832%
16	0	67.623%	>99.999%	>99.999%	>99.999%	>99.999%
16	1	65.188%	>99.999%	>99.999%	>99.999%	>99.999%
16	2	61.762%	>99.999%	>99.999%	>99.999%	>99.999%
16	3	56.673%	99.999%	>99.999%	>99.999%	>99.999%
16	4	49.402%	99.731%	99.878%	99.897%	99.903%
16	5	40.485%	96.404%	97.549%	97.766%	97.868%
17	0	66.395%	>99.999%	>99.999%	>99.999%	>99.999%
17	1	64.145%	>99.999%	>99.999%	>99.999%	>99.999%
17	2	61.030%	>99.999%	>99.999%	>99.999%	>99.999%
17	3	56.704%	>99.999%	>99.999%	>99.999%	>99.999%
17	4	50.474%	99.937%	99.975%	99.979%	99.981%
17	5	42.684%	98.490%	99.084%	99.187%	99.234%
18	0	66.570%	>99.999%	>99.999%	>99.999%	>99.999%
18	1	64.655%	>99.999%	>99.999%	>99.999%	>99.999%
18	2	61.858%	>99.999%	>99.999%	>99.999%	>99.999%
18	3	58.086%	>99.999%	>99.999%	>99.999%	>99.999%
18	4	52.755%	99.990%	99.998%	99.999%	99.999%
18	5	46.004%	99.491%	99.728%	99.765%	99.778%
18	6	37.977%	95.931%	97.170%	97.425%	97.543%
19	0	65.575%	>99.999%	>99.999%	>99.999%	>99.999%
19	1	63.774%	>99.999%	>99.999%	>99.999%	>99.999%
19	2	61.242%	>99.999%	>99.999%	>99.999%	>99.999%
19	3	57.785%	>99.999%	>99.999%	>99.999%	>99.999%
19	4	53.222%	99.998%	>99.999%	>99.999%	>99.999%
19	5	47.222%	99.840%	99.932%	99.942%	99.946%
19	6	40.282%	98.110%	98.806%	98.938%	98.998%

The result of the test program is in the I. The first column is the count of all points of the dataset (denoted n in the header), and the second column is the count of the outliers (denoted o in the header). The table contains all cases where $5 \leq n \leq 19$ and $o \leq \frac{n}{3}$.

The test program has created 70 million test cases in total. The process need around one day in the personal computer of the author.

IV. Conclusion

The recommended method can calculate correct result even in that case the dataset of the control points contains outliers. (If the count of outliers is not too many.) The widely used less squares method produces wrong result, if the measurements contains wrong data.

This method may be useful in every cases, when we would like to create a Helmert transformation from a control point set, which may contain outlier points.

An Python 3 module has been created, which contains the implementation of the recommended method (and the less squares method as a reference). The appendixes contains the source code of this module and an helper module of the test programs. The source code can be downloaded from https://github.com/ngabor/helmert2d under GPL3 license. This code may be use in Python 3 programs, or use as a reference with another applications.

References

[1] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. John Wiley & Sons, 2005, vol. 589.

[2] G. Balkema and P. Embrechts, "Linear regression for heavy tails," *Risks*, vol. 6, no. 3, p. 93, 2018.

[3] S. Choi, T. Kim, and W. Yu, "Performance evaluation of ransac family," *Journal of Computer Vision*, vol. 24, no. 3, pp. 271–300, 1997.

[4] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[5] A. Hast, J. Nysjö, and A. Marchetti, "Optimal ransac-towards a repeatable algorithm for finding the optimal set," 2013.

[6] H. Theil, "A rank-invariant method of linear and polynomial regression analysis," in *Henri Theils Contributions to Economics and Econometrics*. Springer, 1992, pp. 345–381.

[7] G. Nagy, "Sector based linear regression, a new robust method for the multiple linear regression," *ACTA CYBERNETICA*, vol. 23, no. 4, pp. 1017–1038, 2018.

[8] G. Nagy, T. Jancsó, and C. Chen, "The fitting disc method, a new robust algorithm of the point cloud processing," *ACTA POLYTECHNICA HUNGARICA*, vol. 14, no. 6, pp. 59–73, 2017.

[9] F. Edgeworth, "On a new method of reducing observationsrelating to several quantities," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 25, no. 154, pp. 184–191, 1888.

Appendix A

The source code of `helmert.py`:

```
"""Calculate of parametres of Helmert transformation from point pairs"""

from math import sqrt, atan2, sin, cos
from statistics import median
import numpy as np
from numpy.linalg import inv

def trp_helmert(ptl, trp):
    """Calculate Helmert transformation of ptl [(x,y,z), ...]
       by parameters trp (dx, dy, a, b)"""
    return [(trp[0]+pt[0]*trp[2]+pt[1]*trp[3],
             trp[1]-pt[0]*trp[3]+pt[1]*trp[2]) for pt in ptl]

def helmert_lsq(ptl1, ptl2):
    """Calculates parameters from 1 to 2 coordsys, by point lists
       (ptl1 and ptl2 [(x,y,z), ...]), least squeres method"""
    npt=len(ptl1)
    if len(ptl2)!=npt:
        raise ValueError('Different list length (ptl1 and ptl2)')
    prex=(0.0, 0.0, 1.0, 0.0)
    a=np.zeros([npt*2,4], dtype=np.double)
    p=np.identity(npt*2, dtype=np.double)
    l=np.zeros([npt*2,1], dtype=np.double)
    for i in range(npt):
        a[2*i  ][0] = 1.0
        a[2*i  ][1] = 0.0
        a[2*i  ][2] = ptl1[i][0]
        a[2*i  ][3] = ptl1[i][1]
        a[2*i+1][0] = 0.0
        a[2*i+1][1] = 1.0
        a[2*i+1][2] = ptl1[i][1]
        a[2*i+1][3] =-ptl1[i][0]
        l[2*i  ][0] =(prex[0]+prex[2]*ptl1[i][0]+prex[3]*ptl1[i][1]-ptl2[i][0])
        l[2*i+1][0] =(prex[1]-prex[3]*ptl1[i][0]+prex[2]*ptl1[i][1]-ptl2[i][1])
    x=-inv(a.transpose() @ p @ a) @ (a.transpose() @ p @ l)
    return (prex[0]+x[0][0], prex[1]+x[1][0], prex[2]+x[2][0], prex[3]+x[3][0])

def wmed(wml):
    """Wighted median from wml [(weight, vaule), ...]"""
    wmls=sorted(wml, key=lambda element:element[1])
    wmsh=sum(map(lambda element:element[0], wmls))/2.0
    wmsi=0.0
```

```
    for wme in wmls:
        wmsi+=wme[0]
        if wmsi>wmsh:
            return wme[1]

def helmert_rob(ptl1, ptl2):
    """Calculates parameters from 1 to 2 coordsys, by point lists
       (ptl1 and ptl2 [(x,y,z), ...]), robust method"""
    npt=len(ptl1)
    if len(ptl2)!=npt:
        raise ValueError('Different list length (ptl1 and ptl2)')
    dml=[]
    al=[]
    bl=[]
    for ipt in range(npt):
        for jpt in range(ipt):
            c1=complex(ptl1[jpt][0]-ptl1[ipt][0], ptl1[jpt][1]-ptl1[ipt][1])
            c2=complex(ptl2[jpt][0]-ptl2[ipt][0], ptl2[jpt][1]-ptl2[ipt][1])
            cr=(c2/c1).conjugate()
            dml.append( 0.5*(sqrt(c1.real**2+c1.imag**2)+sqrt(c2.real**2+c2.imag**2)) )
            al.append(cr.real)
            bl.append(cr.imag)
    dmed=median(dml)
    wl=[min(dm, 1.5*dmed) for dm in dml]
    a=wmed(zip(wl,al))
    b=wmed(zip(wl,bl))
    dx=median([pp[1][0]-a*pp[0][0]-b*pp[0][1] for pp in zip(ptl1, ptl2)])
    dy=median([pp[1][1]+b*pp[0][0]-a*pp[0][1] for pp in zip(ptl1, ptl2)])
    return (dx, dy, a, b)
```

Appendix B

The source code of `testtool.py`:

```
"""Helper functions for testing transformation methods"""

import random
from math import pi, sin, cos, sqrt

def trcoordlist(ptl1, ptl2, trfunc, olpt=()):
    """Compare the points of ptl1 transformed by trfunc,
and the points of ptl2. Calculate the squared mean with
and without outliers, whose indices are in the olpt.
The result is printes to stdout"""
    trp=trfunc(ptl1, ptl2)
    print('The parameters of the transformation: ',trp)
    sumn=0.0
    sumnall=0.0
    for pp in zip(ptl1, ptl2, [(i in olpt) for i in range(len(ptl1))]):
        x1=pp[0][0]
        y1=pp[0][1]
        x2=pp[1][0]
        y2=pp[1][1]
        x2r=trp[0]+trp[2]*x1+trp[3]*y1
        y2r=trp[1]-trp[3]*x1+trp[2]*y1
        dx=x2-x2r
        dy=y2-y2r
        dr=sqrt(dx**2+dy**2)
        if not pp[2]:
            sumn+=dr**2
        sumnall+=dr**2
        print(f'{x1:9.2f}   {y1:9.2f}   {x2:9.2f}   {y2:9.2f}   {x2r:9.2f}   \
{y2r:9.2f}   {dx:9.2f}   {dy:9.2f}   {dr:9.2f}')
    print(' squared mean: ',sqrt(sumn/(2*(len(ptl1)-len(olpt)))))
    print('with outliers: ',sqrt(sumnall/(2*len(ptl1))))

def randtrp(offs=(-1000.0,-1000.0,1000.0,1000.0),
            scaleiv=(0.999, 1.001), rotiv=(-pi, pi)):
    """Create a random Helmert transformation"""
    scale=random.uniform(scaleiv[0], scaleiv[1])
    rot=random.uniform(rotiv[0], rotiv[1])
    a=scale*cos(rot)
    b=scale*sin(rot)
    dx=random.uniform(offs[0], offs[2])
    dy=random.uniform(offs[1], offs[3])
    return (dx, dy, a, b)
```

```
def randptp(ptn, trp, stderr=0.07071, pt1win=(0.0,0.0,1000.0,1000.0),
            oln=0, oldst=(1.0,10.0)):
    """Create a list of ptn random point in pt1win area (ptl1),
the transformed point by a Helmert transformation of trp (ptl2),
and the random modified points by stderr normal variate error (ptl2e).
oln points will be outlier, whose indices are in olpti.
The function returns (ptl1, ptl2, ptl2e, olpti).
"""
    ptl1=[(random.uniform(pt1win[0], pt1win[2]),
           random.uniform(pt1win[1], pt1win[3])) for i in range(ptn)]
    ptl2e=[(trp[0]+trp[2]*pt[0]+trp[3]*pt[1],
            trp[0]-trp[3]*pt[0]+trp[2]*pt[1]) for pt in ptl1]
    ptl2=[(pt[0]+random.normalvariate(0,stderr),
           pt[1]+random.normalvariate(0,stderr)) for pt in ptl2e]
    olpti=random.sample(range(ptn),oln)
    for oi in olpti:
        ptx,pty=ptl2[oi]
        ptx+=random.uniform(10.0,50.0)*random.choice([-1,1])
        pty+=random.uniform(10.0,50.0)*random.choice([-1,1])
        ptl2[oi]=(ptx, pty)
    return (ptl1, ptl2, ptl2e, olpti)

def trpstat(ptl1, ptl2, olpt=[]):
    """Calculate the squared mean and the mean of the absolute values
of the difference between the points of ptl1 and ptl2 lists.
The calculation skipts points, whose indices are in the olpt.
The function returns (squared mean, absolute mean) in a tuple."""
    nn=len(ptl1)
    n=nn-len(olpt)
    sumr2=0.0
    sumra=0.0
    for ptp in zip(ptl1, ptl2, range(nn)):
        if not ptp[2] in olpt:
            dx=ptp[1][0]-ptp[0][0]
            dy=ptp[1][1]-ptp[0][1]
            dr2=dx**2+dy**2
            sumr2+=dr2
            sumra+=sqrt(dr2)
    return (sqrt(sumr2/n), sumra/n)
```